2011 First ACIS/JNU International Conference on Computers, Networks, Systems, and Industrial Engineering

A Design of the Conceptual Architecture for a Multitenant SaaS Application Platform

Sungjoo Kang[1], Sungwon Kang[2], Sungjin Hur[1]
Software Service Research Team,
Electronics and Telecommunications Research Institute[1]
Department of Computer Science,
Korea Advanced Institute of Science and Technology[2]
E-mail: sjkang@etri.re.kr

Abstract

In this paper, the conceptual architecture of a SaaS platform that enables executing of configurable and multitenant SaaS application is proposed. Even though most of organizations utilize a standardized SaaS application developed by a SaaS developer, each of tenants would have unique requirements regarding service components. The platform provides several features for configuring these aspects of SaaS software such as organizational structure (role sets and access control), user interface, data model, workflow, and business logic. To satisfy multitenancy of SaaS application, we applied metadata driven architecture proposed by Force.com. The target SaaS platform is composed of the key components that supports SaaS Application execution environment that serves multiple tenant using a single service instance. An example scenario of SaaS software lifecycle is also described to explain how the target platform would be operated from development and deployment of SaaS application to configuration by tenants.

1. Introduction

Software-as-a-Service (SaaS) is a novel delivery model of software that provides an application for multiple users via internet as a form of 'ondemand service' [1]. By subscribing SaaS service, companies can use various IT services without the need to purchase and maintain their own IT infrastructure [2]. Moreover, the SaaS service provider can offer SaaS services at a moderate price by making full use of the economy of scale.

The maturity of SaaS service can be defined by several maturity models [3,4]. Three key attributes – Configurability, Multitenancy, and Scalability are widely used in these models. Microsoft proposed SaaS Simple Maturity Model [3] which described maturity of SaaS architecture with four maturity levels. The lowest level (Level 1) represents the Application Service Provider (ASP) model [5] which requires a dedicated server and service instance for each tenant (i.e. a company subscribing the service). At this level, maintaining cost of the service provider is high because it requires multiple different instances for different tenants. Configurability is achieved at the Maturity Level 2. Although it still requires dedicated server for each tenant, identical instances can be used because it provides a feature for tenants to configure some aspects of SaaS software as they want. The Maturity Level 3 provides Multitenancy which represents the ability that enables SaaS application to serve multiple tenants using a single service instance. At this level, tenants not only have feature to configure some aspects of SaaS software such as UI or data model but also feels as if they are using a dedicate server while every tenant are sharing a single server and service instance. In terms of operation cost, the benefit of economy of scale can be achieved by the Maturity Level 3. At the highest maturity level, scalability is added through a multi-tier architecture with a load-balancing feature. The system's capacity can be increased or decreased by adding or removing servers.

Several researches were carried out to study configurability of SaaS application. Nitu [6] classified several configurable aspects of SaaS software and proposed SaaS application architecture. In this architecture, several kinds of aspect - user interface, workflow, data, and access control could be configured by the tenants' designer and configured data are stored as configuration data which is used to generate customized application in runtime. However the architecture mainly focused on configurability of the application and multitenancy was not discussed well.

Well modeled configurability was regarded as the key characteristic to satisfy multitenancy of SaaS application. Mietzner et al. [7] introduced the concept of software product lines engineering to model variation points of SaaS application at development stage. After deploying application at production stage, modeled variability of application can be configured by different tenants while maintaining service instance as a single product family. However, the architecture for implementing this concept was not proposed.

To extend configurable limit of SaaS application, customization which involves SaaS application source

978-0-7695-4417-5/11 $26.00 © 2011 IEEE
DOI 10.1109/CNSI.2011.56

code change was issued. Sun et al. [8] clarified the difference between configuration and customization. They proposed a competency model and a methodology framework to help SaaS vendors to plan and evaluate capabilities and strategies for service configuration and customization.

The multitenant architecture with single instance (Maturity Level 3 or above) was implemented by Salesforce.com [9] by introducing their metadata driven architecture and APEX programming environment. However, it is hard to be implemented by public since it is developed using on-premise technology.

In this paper, we propose a conceptual architecture of SaaS platform that provides configurable and multitenant SaaS application. For achieving high degree of configurability, five configurable aspects of SaaS software and the way to achieve it are discussed in section 2. For providing multitenancy, several required components and their mechanisms to generate polymorphic application at runtime are dealt with in section 4. In section 5, the conceptual architecture is evaluated with execution scenario.

2. Configurability of SaaS Application

In this section, we will define two kinds of SaaS application that would be provided by the SaaS platform. Several configurable aspects of these applications would be introduced to define configurability of the platform. Finally, metadata driven architecture would be explained to show how multitenancy is satisfied.

2.1. SaaS Application

The SaaS application operating on the proposed SaaS platform is one packaged business application with web-based user interface to multiple tenants. The purpose of business application such as CRM, ERP, or Groupware is processing business transactions and collaboration among tenants' users with business data in DBMS as the center. The three tier architecture as in Figure 1 is widely used for operating business application.

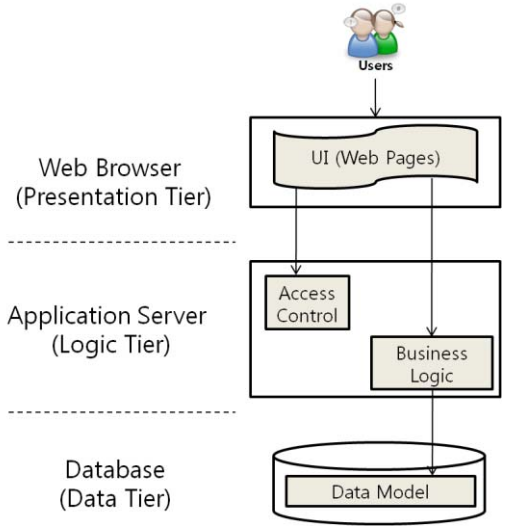

Figure 1. Execution of business application

Data objects for a business application are defined in Database Management System (DBMS). Different users in a tenant have different roles such as boss, manager, or employee and every role set have their own authorities for accessing company data and business logics. Based on role sets and their access authority, web pages are provided for users to request services through web browsers. Requests are transferred to the Application Server which contains business logics that processes business transactions. The main aspects of a business application consist of data model, business logic, web pages, and role sets.

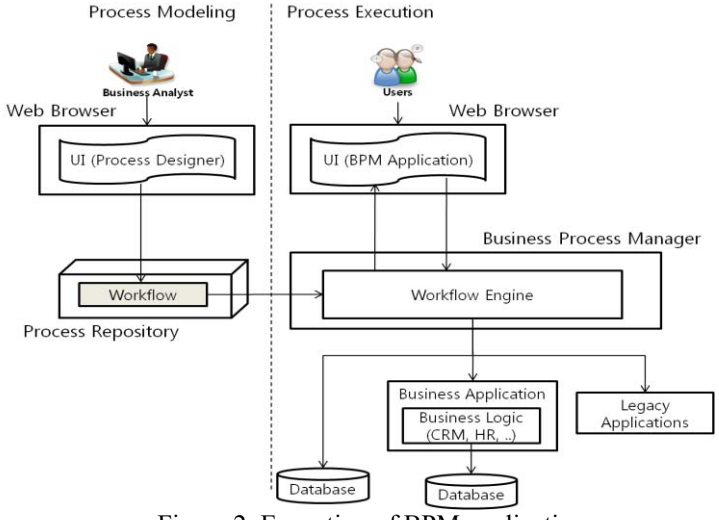

Figure 2. Execution of BPM application

In addition to business applications, Business Process Management (BPM) application is provided by the proposed SaaS platform. In contrast with other business applications that are used for sharing information and collaboration among users, BPM application focuses on the flow of related activities of an organization. Figure 2 illustrates the execution of the BPM application.

The purpose of BPM applications is modeling and executing business processes, which are collections of related, structured activities of an organization. Business process is modeled by a business analyst or a manager of the organization. The modeled business process is stored as a form of a workflow in the Process Repository. A workflow is a depiction of a sequence of operations which executes modeled business process using a workflow engine. Each member of the organization has a user interface to check and to process activities assigned to the member. The main aspects of a BPM application are workflow and its activities.

2.2. Configurable Aspects of SaaS Application

Even if most of organizations subscribe a standardized SaaS application deployed on the platform, each of tenants would have unique requirements regarding service components such as organizational structure (role sets and access control), user interface, data model, workflow, and business logic. The following subsections describe configurable aspects of SaaS application.

2.2.1. Organizational Structure. Configurability of an organizational structure is the first essential feature because it is a very unique aspect of tenants that can be often changed and it is impossible to consider them at the development stage. A tenant's (configuration) manager can add or delete role sets based on initial roles that SaaS application developer created. The tenant manager also can set a number of authority levels for accessing data model and assign them to each role sets in the organization. After the tenant manager sets a certain authority level for a role, the SaaS platform checks the authority when data model is accessed or service is requested by the tenant user. For example, company's secure data can be accessed by executives but not by employees. Data that are not opened to suppliers are not seen through supplier's UI page. Some of workflow such as payroll process cannot be activated by a support division member.

2.2.2. User Interface. Configuration of user interface is to change look and feel of the UI. Tenant manager can change colors of component in UI pages, restructure page layouts, add/delete hyperlinks, or add UI component to show additional data model. Moreover, it is possible to differentiate service components based on different role sets of an organization. Meanwhile, it is important to consider the impact of the change of a component to other aspects. For example, if a new data field is added to any data object, a new column is added in a grid UI component that shows fields of the data object. Similarly, if any data object is deleted from the database, UI page for manipulating the data object is also to be inaccessible.

2.2.3. Data Model. Configuration of data model means the ability to add/delete data object, to add/delete data fields in existing data object, and to change the field name, type, and so forth. Tenant manager can access DBMS of the platform and manage data model of SaaS software. Additionally, the tenant manager can configure access authority of role sets for each data model and data field based on defined authority levels.

2.2.4. Workflow. Since different organizations have unique organizational structures, decision making process, or business rules, workflows designed by application developer should be configured by each tenant. Using the Process Designer tool, the tenant manager can configure various components of business process in terms of workflow, activity type, and business rules.

2.2.5. Business Logic. Business logic is main functionalities of business applications that handle information exchange between a database and a user interface. For example, core business logics of CRM application are 'get list of all customers information', or 'add all sales together' and of email application are 'reply email to sender', or 'list address information'. Therefore, fully tenantized business configuration can transubstantiate the purpose of the original application. Nevertheless, we sometimes are asked the tenant's requests to add new business logic. In this case, tenant manager can create a simple business logic using template class. This class provides several methods to manipulate data model and to return results of arithmetic operations on data in a database. Then, the generated business logic is dynamically loaded on the platform and being serviced via tenant specific URL. Then, the tenant manager creates new UI page and associates with the page and the URL.

2.3. Multitenancy via Metadata

Since we provide SaaS application for multiple tenants with single service instance, the platform architecture needs to enable self configuration by tenants without changing the SaaS application source code for individual tenant and runtime configuration not to suspend service during the configuration. Metadata driven architecture by Salesforce.com [9] provides solution for self- and runtime configuration of SaaS application. Figure 3 shows the concept of metadata driven architecture.

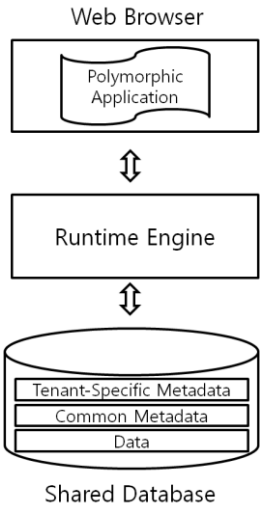

Figure 3. Metadata driven architecture

In this architecture, every aspects of SaaS application that are configurable by tenants are stored in metadata database. When a tenant manager configures some aspects of SaaS application, configured information is stored separately in tenant-specific metadata. Runtime engine generates polymorphic application for individual tenant using application codebase and tenant-specific metadata at runtime. Through the polymorphic application, tenant users feel as if they are using their own business application while service instance is shared by every tenant. Even though application data are shared, they are kept secure because the polymorphic application for individual tenant accesses application data independently via optimized query for each tenant. In this paper, we applied metadata driven architecture to achieve multitenancy with single service instance. While Salesforce.com developed their on-premise solution, we adopted open source software.

3. The Conceptual Architecture

The proposed SaaS platform provides SaaS Application execution environment that serves multiple tenant using a single service instance. To do this, the platform is composed of several key components – configurator, runtime engine, metadata management system, and so on. Figure 4 depicts the conceptual platform architecture for the SaaS platform.

Detailed descriptions of the components of the target platform are given in following subsections.

3.1. Client application

Tenants can use SaaS services through a web browser. Enterprise portal provides a workflow application and interfaces to various business applications. When the web browser sends HTTP request, the Tomcat Application Server provides the web application developed in ExtJS based Javascript library and responds to JSON request of the web application.

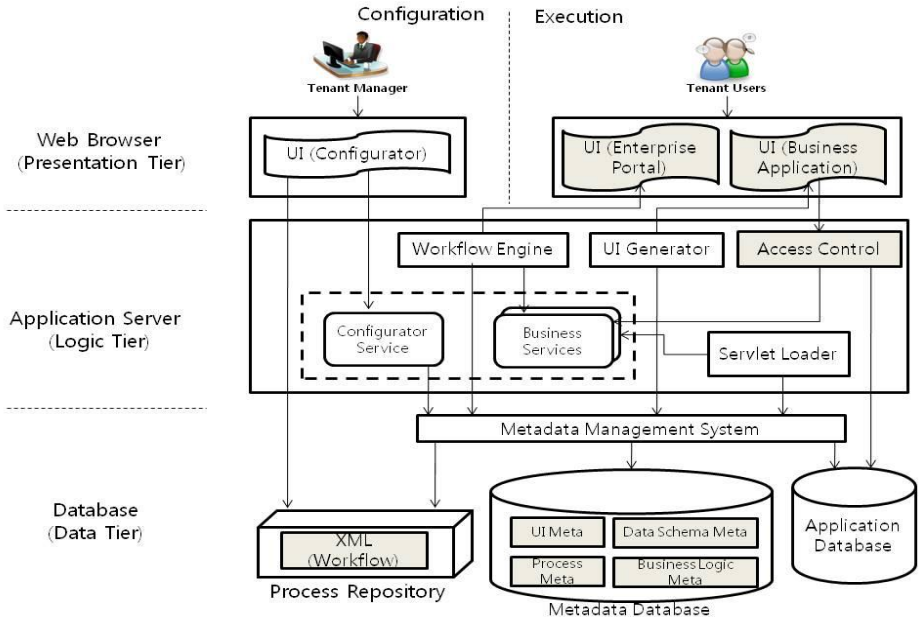

Figure 4. Conceptual architecture for SaaS platform

3.2. Configurator

Configurator is an ExtJS based web application used by the tenant manager for configuring various aspects of SaaS application. Configurable aspects in this platform are UI pages, organizational structure of the tenant, data models, workflows, and business logics. Configurator provides a drag-and-drop interface to configure a given web page by arranging ExtJS containers and components in the web page. Moreover, tenant manager can create new web page with various ExtJS containers and components. In the platform, each container and component has unique ID. IDs of container and component that compose a certain web page are stored as UI metadata with a page. When the web page is requested by a tenant, the web page is dynamically generated with the metadata and associated application data.

Configurator provides a graphical user interface to configure data model and organizational structure and access authority. Using the configurator, like a database client program, the tenant manager not only is able to view data object and its data fields but manipulates their information and access policy.

Configuration of workflows and business logics can be done by a design interface of the configurator. Using the interface, the tenant manager can modify a workflow with several features – rearranging order of activities, changing assigner of an activity, and so on. The tenant manager can compose new workflow with predefined activity types.

The tenant manager can add new business logic with the designer as well. One of the activity types is performing database operation. The tenant user can create new business logic by composing a simple workflow with a database operation that processes business logic tenants need. After composing the business logic, it is compiled and deployed in the platform with an accessible URL. By the UI configuration feature, the tenant manager can make new UI page for the new business logic.

3.3. Runtime Engine

The SaaS application that operates on the proposed SaaS platform is one packaged business application with web-based user interface to multiple tenants. The purpose of business application such as CRM, ERP, or Groupware is processing business transactions and collaboration among tenants' users with business data in DBMS as the center. The three tier architecture is promoted for operating business application as Figure 1. Configured aspects of SaaS application by the tenant manager are stored as metadata in the metadata database. While, codebase developed by the application developer is stored in the application database. Runtime engine plays a role in generating tenant specific application using codebase and metadata. Followings are subcomponent of runtime engine:

- UI Generator: The metadata of a web page is composed of IDs of ExtJS container and components. When the page is requested, UI Generator builds a tenant specific UI page in runtime with the metadata in the UI Meta DB.
- Workflow Engine: Workflow Engine parses and executes Business Process and Rules. When a workflow activated by a user, the Workflow Engine retrieves the workflow from the Process Repository and executes activities in sequence from the Process Meta DB.
- Servlet Loader: The tenant generated business logic (compiled Java class) is stored in the Business Logic Meta DB. When the business logic requested by an UI component, Servlet Loader retrieves the Java Class and loads it into the platform as a Servlet dynamically.

3.4. Metadata Management System

Metadata Management System provides two key features for supporting multitenancy. The first one is an access control for supporting multiple tenants. When a developer makes business logic in a SaaS application, he cannot help making complicated SQL query with tenant's ID and tenant configured data object and field name. To avoid this painful task, Metadata Management System provides metadata APIs for Logic Tier. By utilizing metadata API, Logic Tier can access shared database regardless of tenant's information. When the web browser sends request to the Application Server, business logic access to database with metadata API. Then, Metadata Management System converts the request to the optimized query to retrieves tenant specific UI pages and data. It provides secure and independent data access to shared database for each tenant.

The other feature is providing extended fields for data object. At development stage, it is impossible for the developer to predict what data models and data fields the tenant manger added in the future. In this platform, therefore, ten fields are added to data model when the data model created. However, the usage of these extended fields is different for each tenant. For example, some of tenants would use three of ten extended fields for *customer* data model but others would use nothing. Another tenant would use the first extended field as *logined_at* field to *customer* data model as Date type but the other would use it as *phone_number* as Varchar type. Therefore, information of extended fields for every tenant should be managed in Metadata Management System. This information is used for retrieving tenant specific UI pages in runtime.

3.5. Database and Repository

There are two kinds of database in the platform. Metadata database stores configured aspects of SaaS application by the tenant manager. Application Database manages codebase of applications and user generated data. XML-based workflow files are stored in Process Repository.

4. An Example Scenario

Figure 5 illustrates a diagram showing interactions among actors and the SaaS platform. In this scenario all phases of the lifecycle of a business application are described. At first, an application developer creates codebase of a business application through SaaS SDK. After deploying the application for tenant companies, a tenant manger in the company can configure five different aspects of the application through Configurator Service. Since the original codebase developed by the application developer is used for multiple tenants, configured aspects of the application by the tenant manager are stored for each tenant separately from codebase.

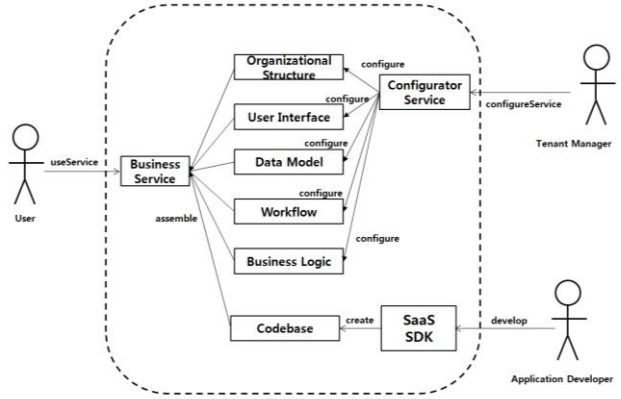

Figure 5. Saas Platform Interaction Scenario

Users in the tenant company access the application through Business Service. When the users request the service, the business service is generated and loaded by assembling codebase and configured aspects of the application if there are any aspects of the application configured by their tenant manager.

Through the SaaS platform interaction scenario, we can check the functionality of the key components and the satisfaction of configurability and multitenancy of the platform.

5. Conclusion

In this paper, we proposed a conceptual architecture of a SaaS platform that enables executing of configurable and multitenant SaaS application. The platform allows the configurator application to configure five aspects of SaaS software. In addition, metadata driven architecture composed of Runtime Engine, Metadata Management System, and Metadata DB are applied for providing multitenancy of SaaS application.

We plan to implement the platform based on the conceptual architecture. Furthermore, we are going to provide Software Development Kit (SDK) for SaaS application development that deals with commonality and variability model of aspects of SaaS application using the software product line approach.

6. Acknowledgement

This work was supported in part by the IT R&D program of MIS/IITA [2009-S-033-01, Development of SaaS Platform for Software Service of Small and Medium Enterprises, and Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology [20100022431].

7. References

[1] Software as a Service, *http://en.wikipedia.org/wiki/-Software_as_a_service*

[2] Cor-Paul Bezemer and Andy Zaidman, "Multi-Tenant SaaS Applications: Maintenance Dream or Nightmare?", *Technical Report (TUD-SERG-2010-031)*, Delft University of Technology

[3] Gianpaolo Carraro, "Understanding SaaS Architecture: A Simple SaaS Maturity Model," *http://msdn.microsoft.com/enca/architecture/aa699384.aspx, 2006*

[4] Stephan Ried, "Forrester's SaaS Maturity Model: Transforming Vendor Strategy while Managing Customer Expectations," http://*www.forrester.com/Research/-Document/Excerpt/0,7211,46817,00.html*, 2008

[5] Application Service Provider, *http://en.wikipedia.org/wiki/Application_service_provider*

[6] Nitu, "Configurability in SaaS Applications", *2nd India Software Engineering Conference*, 2009, pp. 19-26.

[7] Ralph Mietzner, Andreas Metzger, Frank Leymann, and Klaus Pohl, "Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications", *In Proc. of the ICSE Workshop on Principles of Eng. Service Oriented Systems (PESOS)*, 2009, pp. 18-25.

[8] Wei Sun, Xin Zhang, Chang Jie Guo, Pei Sun, Hui Su, "Software as a Service: Configuration and Customization Perspectives", *In Proc. of the IEEE Congress on Services Part II*, 2008, pp. 18-24.

[9] Craig D. Weissman and Steve Bobrowski, "The design of the force.com multitenant internet application development platform", *In Proc. of the SIGMOD*, 2009, pp. 889-896.