Web-based Development Framework for Customizing Java-based Business Logic of SaaS Application

Jihyun Lee, Sungju Kang, Sung Jin Hur
Cloud Computing Research Department, ETRI, South Korea
{jihyun, sjkang sjheo}@etri.re.kr

Abstract—
Tenants lease business applications in form of SaaS (Software as a Service) and their administrators are involved in only adding and modifying the data-processing code in order to operate their specific logics by using a web-based development framework. Especially, customizing of business logics of SaaS applications makes a tenant administrator easy and rapid development by focusing on the data-processing code without concerning about the designing application, administrating database system, and making other additional implementation code. Therefore, the issues of customizing business logics are interesting and challengeable in the SaaS domain. In this paper, we propose the web-based development framework to generate custom business logics of SaaS applications and the runtime engine to process the changes based on the multi-tenant execution environment. The generating and executing custom business logics is core feature and scheduled to be integrated with the SaaS platform, SaaSpia^{TM}, which has been developed in ETRI from 2009.

Keywords— **SaaS platform, customization, business logic, multi-tenant**

I. Introduction

Software as a Service (Software as a Service) is a one-to-many software delivery model in which both application and host environment are centrally managed and shared with several companies who lease and use application services [1].

Independent Software Vendor (ISV) have sold software targeting mass marketing, however, they have specialized to provide specialized applications and services for a particular company's requirements. In the concept of SaaS, Service Delivery Platform (SDP) allows economy of scale by providing services based on "single instance multi-tenancy" [2]. Furthermore, diverse platforms in the type of an infrastructure have been existed until now. However, in the perspective of reflecting diverse requirements, these platforms fall short of multitenancy supportive customization capability.

More requirements from enterprise companies are making SDPs to offer multitenancy and customizability [3][4][5], which make the new economies of scale. We investigate the addition of instance-level information to generate business logic code at runtime and the on-the-fly execution of business logic codes. Our suggested approach is included in the scope of the scheduled progress until 2011.

This paper described multitenant supportive customization architecture, in addition to instance-level business logic customization at runtime and execution. This paper consists of the following sections: Chapter 2 describes SaaS customization architecture. Chapter 3 proposes the instance-level business logic customization based on the SaaS hosting platform foundation. In Chapter 4, related works are described and we conclude in Chapter 5.

II. SaaS Customization Architecture

We define that business logics of the specific web page (or web site) is necessary functionality to provide dynamic interactions and related services to users. However, once SaaS services are deployed on their hosting platform, it is hard to modify implementation-level business logics even without stopping running hosting platform.

Especially, SaaS SDP is oriented one-instance based multitenancy and it does not allow stopping and re-starting the web server shared by multitenants [6][7]. Therefore, SaaS SDP should have a solution to generate and modify instance-level customization information, which mean code-based customization of logics and data. In this paper, we investigated how to configure customization information about new and updated business logics and execute the logics from the user interfaces.

For tenant's satisfaction, the runtime code generation and execution is meaningful especially in SaaS SDP because it can support to make not yet implemented code available in runtime platform on the fly. Furthermore, our suggested approach allows multitenants to modify and generate Java based business logics and make relationship with the specific UI component in order to be invoked from that UI component.

Figure 1 shows the architecture of SaaS platform in the perspective of service customization and it is composed of the modules named as Request Dispatcher, UI Renderer, Configurator, Runtime Execution Engine, and Repository.

The Request Dispatcher is used to pass the current request to another page for service requests. The UI Renderer reads the structured components and configuration information, and display new pages by interpreting it. With the Configurator module, a tenant administrator can describe his/her customization requirements to modify UIs, business logics, and data schemas of services.

The Runtime Execution Engine provides the facilities such as dynamic code generation, compilation, and loading at runtime. Therefore, custom business codes can be dynamically generated at runtime and to be executed for execution requests.

The Repository stores information about tenants, configuration actions, and core applications shared by multi-tenants and provides functions of retrieval of the stored data to each tenant.

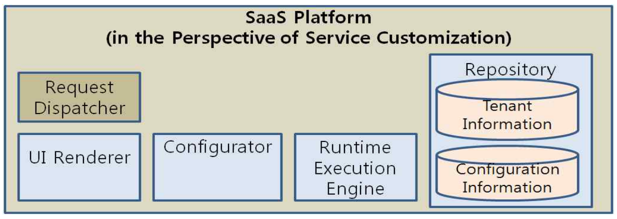

Figure 1. Architecture of SaaS Customization Platform

III. Instance-Level Business Logic Customization

Until now, we have shown the customization support of SaaS platform. The function of customization makes the shared core SaaS application to be modified adequately for the use by multitenants. In the SaaS customization, the important artifact is metadata because metadata are generated and stored as customization information of each tenant.

In this paper, we describe how to generate new business logic in runtime and update the logic to the hosting platform on the fly. For the customization of instance-level business logic, we classify the business logic into three kinds of metadata.

A. Classification of Business Logic Metadata

1) SQL DML statement: SQL statements especially for data manipulation can be re-defined. Therefore, the query statement deals with commands for data manipulation through select, insert, update, and delete queries. The select statement is used to retrieve rows from the database and has the most complicated structure among other SQL statements. When a tenant administrator customizes SQL queries, he/she should know which databases, tables, and columns are used to manipulate data.

2) Logic code: After defining a SQL statement, tenant administrators can develop Java-based business logics in which the SQL statement is embedded. In the logic code, program code is defined and the value retrieved from execution of queries can be manipulated and stored in host variables. The key and value pairs are returned and the returned values will be represented.

3) ***Logic script***: Through a logic script, the simple operations such as sum, average, and arithmetic operation can be defined with JavaScript and also executed. Compared to Java-based logic code, JavaScript is interpreting language; therefore, it does not need to be compiled and re-loaded in the hosting platform.

Business logics can have two types: one is the SQL statement combined with logic code and the other is the logic script without a SQL query. In this paper, we focus on the first type, that is, the combination of SQL statements and logic codes.

B. Customization of Business Logics

1) ***Operation definition***: For operation definition, SaaS platform provides tenant administrators with simple operators and templates for developing custom operations. The simple operators are for arithmetic, sum, average, maximum, and minimum operations. Furthermore, if tenants want to define the specific operation which cannot handle with the simple operators, they can finally define business logic of the operation in Java.

2) Connection to UI component: The defined business logic as a code is associated to the specific UI component in the web page and it is executed when the service is requested. Consequently, through this execution process, the UI component can get values to be shown and also display the sent information after post-processing in the side of the UI component itself.

3) *Storing metadata*: Once business logic is defined, it can be stored and re-stored after being re-defined when tenant administrators desire to define new business logic.

Figure 2 shows how business logic metadata to be defined (annotated as 1 and 2 in Figure 2) and associated with a specific UI component (in the number 3) step by step.

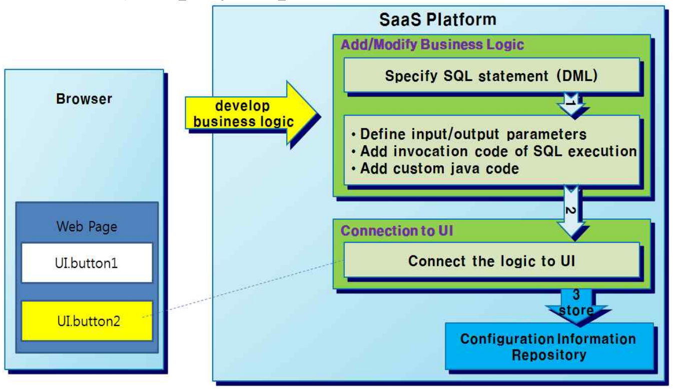
SaaS Platform
Browser
develop business logic
Add/Modify Business Logic
Specify SQL statement (DML)
1
• Define input/output parameters
• Add invocation code of SQL execution
• Add custom java code
2
Connection to UI
Connect the logic to UI
Web Page
UI.button1
UI.button2
3
store
Configuration Information Repository

Figure 2. Generation of Metadata-based Business Logic

C. Execution of Business Logics

1) Retrieval of business logic metadata: When the execution of service's operation is requested from the browser by a service user, metadata of business logic are retrieved from the configuration information repository. At this point, metadata retrieval is only for the specific tenant and other tenants cannot access to other tenants' metadata.

2) Metadata-based code generation: The retrieved metadata for the query statements, code-based logics, and key-value pair collections are combined into a source code template. The described information of the template becomes new source code.

3) On-the-fly compilation and loading: Based on the retrieved metadata, new source code is dynamically generated. After the generated source code is automatically compiled by dynamic compiler, the compiled class file is loaded in the web server.

4) *Execution of service operation*: And then, the service request is processed by invoking operations defined as the metadata-based business logic.

Figure 3 shows internal processing such as metadata retrieval (shown in the number 1 and 2), metadata-based code generation (in the number 3), loading in the SaaS platform after on-the-fly compilation (in the number 4), and execution by invoking business logics (in the number 5). The return values as the result of the specific logic from the execution are displayed in the web page (in the number 6) and consequently they can be shown to end users of the services. During the process of executing business logics, end users of the services are not necessary to know about the following tasks such as metadata retrieval, source code generation, dynamic compilation and re-loading to the hosting platform. The only task of the end users is just requesting service from their browser.

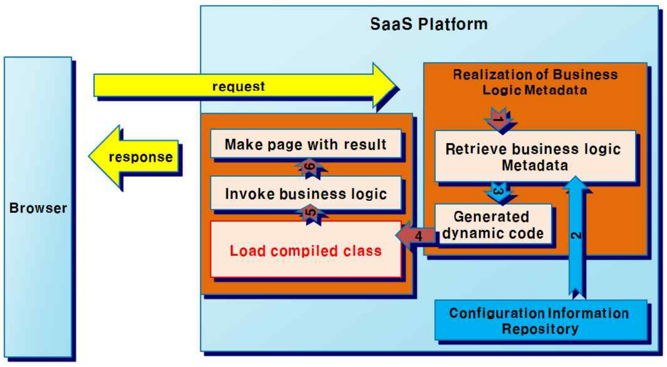

Figure 3. Execution of Metadata-based Business Logic

IV. Related Works

The Composite Application Framework (Cafe) [8] suggested the generic configurable customizable framework for generating composite cloud application by integrating configured application components with infrastructure components. The Cafe distributes and deploys web-services with customizable components of a cloud application. However, the components are formed as SOA-based web services. Tenants can show components just as a black box; therefore, if the tenants want to modify content and layout, the Cafe cannot support to modify the internal elements such as business logics and outputs.

Web applications are generally data-centric services [9] [10] and they handle with queries and logics. Therefore, to design of queries is important to the tenant, if necessary; the tenant wants to add and change queries and also define the result sets of the database queries [11]. However, the customization of on-demand software services, the most cases are working with both SOA-based web services and business process. In the case of business process, customization and composition are processed to make model of tenants' personalized business activities by depicting inputs, outputs, pre-conditions, effects, and QoS [12]. Therefore, the customization is in the level of composing interfaces and adapting inputs and outputs.

The metadata based customization [13] provides the improvements of metadata-based architecture for supporting the dynamic customization. The metadata information in SaaS customization represents the changes that have been made to the data objects to compose SaaS application of the parts during the customization process. And then customization metadata can be applied to the component elements of the whole application.

In this paper, we propose the generic framework to customize code-level business logics and database queries while it still provides support of the properties such as multitenancy supporting customizability and in the code and query level granularity.

Our contributions are as follows: firstly, it is possible to change code-level content by (re-) defining database queries and Java-based codes even though SaaS platform is running. The customization information can be reflected to the platform without stopping and restarting the platform itself in runtime. Secondly, we propose the customization architecture which is separated into functionality and a layout. Through this architecture, tenants can separately modify layout, content information to be identified from the database, and display outputs.

V. Conclusions

In this paper, we investigate the method that each tenant can customize the platform-based services, especially business logics by generating new source code with metadata in runtime. The modified logic can be executed when users request. Furthermore, even though the execution environment is being shared by multi-tenants, the customized modules can be separately stored and dynamically reflected only to the users of the tenant that specifies customization information.

The runtime execution of customized business logic included in the SaaS customization platform is in the scope of our project in this year. The facilities such as dynamic code generator, on-the-fly compiler, and dynamic loader are being developed. The configurator module in our SaaS platform was developed for specifying the metadata of code-based business logics through web browser. Integration of these modules with our SaaS platform is scheduled in the end of this year.

References

[1] Peilin Guo, “A Survey of Software as a Service Delivery Paradigm,” TKK T-110.5190 Seminar on Internetworking, 2009.

[2] Gianpaolo Carraro, Fred Chong, and Eugenio Pace, “Efficient Software Delivery through Service-Delivery Platforms,” The Architecture Journal, MSDN, 2011.

[3] Deshuai Wang, Yichuan Zhang, Bin Zhang, Ying Liu, "Research and Implementation of a New SaaS Service Execution Mechanism with Multi-Tenancy Support," ICISE 2009, pp. 336 - 339, 2009.

[4] Craig D Weissman, Steve Bobrowski, "The Design of the Force.com Multitenant Internet Application Development Platform," Int. Conf. on Management of Data, pp. 889-896.

[5] Javier Espadas, David Concha, Arturo Molina, "Application Development over Software-as-a-Service platforms," ICSEA 2009, pp.97 - 104, 2009.

[6] Cor-Paul Bezemer, Andy Zaidman, "Challenges of Reengineering into Multi-Tenant SaaS Applications, Technical Report, TUD-SERG-2010-012, 2010.

[7] Thomas Kwok, Ajay Mohindra, "Resource calculations with constraints, and placement of tenants and instances for multi-tenant saas applications," ICSOC 2008, pp. 633-648, 2008.

[8] Ralph Mietzner, Tobias Unger, Frank Leymann, "Cafe: A Generic Configurable Customizable Composite Cloud Application Framework," Lecture Notes in Computer Science, Volume 5870/2009, pp. 357-364, 2009.

[9] Bernstein, P.A., Melnik, S., "Model Management 2.0: Manipulating Richer Mappings," VLDB 2007, pp. 1-12, 2007.

[10] Clemens Kerer and Engin Kirda, "Layout, Content and Logic Separation in Web Engineering," WebEngineering 2000, LNCS 2016, pp. 135–147, 2001.

[11] Jianwu Wang and Jian Yu, "A Business-Level Service Model Supporting End-User Customization," ICSOC 2007, pp. 295-303, 2007.

[12] Yuliang Shi, Shuai Luan, Qingzhong Li, Haiyang Wang, "A Flexible Business Process Customization Framework for SaaS," ICIE 2009, pp. 350 - 353, 2009.

[13] David Hicks, Klaus Tochtermann, Thomas Rose, and Stefan Eich, "Using Metadata to support customization," Proceedings of the Third IEEE Metadata, 1999.