

Multi-tenant Supporting Online Application Service System Based On Metadata Model

Byeong-Thaek OH*, Hee-Sun WON*, Sung-Jin HUR*

* ETRI(Electronics and Telecommunications Research Institute), Korea

btoh@etri.re.kr, hswon@etri.re.kr, sjheo@etri.re.kr

Abstract— In this paper, we present an online application service system which includes a SaaS platform server. We applied code generation and metadata model technology in order to serve multiple tenants with a single instance of web based online application.

Keywords— Single-Instance Multi-tenant, Metadata, SaaS, Software-as-a service, Online configuration

I. INTRODUCTION

In the past, software had to be installed in an infrastructure close to end users. Since early 90's there has been a rapid advance in the internet in terms of speed, connectivity and reliability. With the internet becoming ubiquitous, coupled with the vendor's interest in capturing the market of small customers who could not afford the expensive enterprise software, either on-premise or through Application Service Provider (ASP), fuelled the advent of Software as a service (SaaS). SaaS is a software delivery paradigm where the software is hosted off-premise and delivered via web and the mode of payment follows a subscription model. At this time SaaS model seemed very similar to ASP model, but there were many subtle differences between the two. ASPs hosted third party client server applications in a data center kind of environment. Since each ASP had many customer-specific applications running, it couldn't provide much expertise in each application. Customers still had to have in-house expertise to make sure the applications were operated correctly. Also building and maintaining data centers was a costly affair for the ASPs and the high cost filtered to the customers. A typical SaaS provider offered applications specifically designed to be hosted and delivered over the Internet to many customers. With SaaS, the customers' IT people had access to a number of different features or capabilities within the software set and could configure them via a web interface to fit the needs of their company. A true SaaS model adhered to the multitenant architecture, in which all users and applications share a single, common infrastructure and code-base that is centrally maintained. The fine configuration allowed this single-instance multiple-tenant model. Thus, even though there was single instance of the software running, the customers got a unique application depending on the configuration done by them. Because SaaS vendor clients are all on the same infrastructure and code-base, vendors can innovate or update the application more quickly

and save the valuable development time previously spent on maintaining numerous versions of outdated code[6]. And also, since SaaS applications were centrally managed by the vendor, there was no need for in-house application developers or experts at customers end. The current industry-wide trend is for Internet based services. Deployment of software as a service, accessible on the Internet and supported by multi-tenant architecture, makes new applications (or tenants) available with significantly lower costs. In this paper, we present an online application service system which is applied code generation and metadata model technology in order to serve multiple tenants with a single instance of web based online application.

II. RELATED WORK

Most software is designed to be deployed on-premises prevailing application platforms, such as Java EE and .NET, have been designed to meet this use model. For SaaS providers, the underlying computing resources must be shared among tenants without interference from other tenants. Vendors intending to compete as providers of application platforms as a service (APaaS) should prepare to use (their own or third-party) multitenant SaaS/cloud-enabled application platform (SEAP) technology

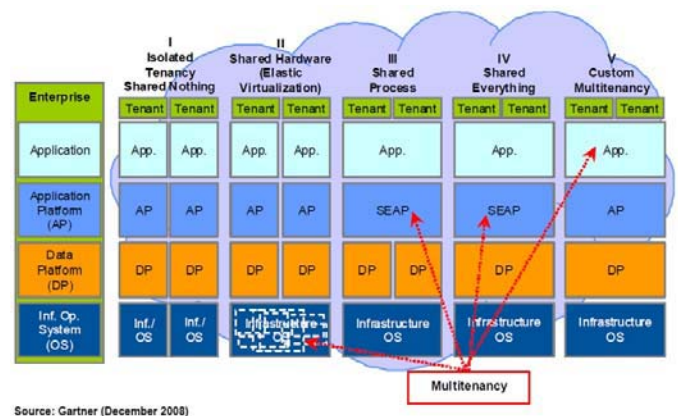


Figure 1. Gartner Reference Architecture for Multitenancy[1]

Figure 1 shows the reference architecture for multi-tenancy. On the figure, the model I, each tenant has its own dedicated stack; nothing is shared. This is not true multi-tenancy, and

this model is similar to traditional hosting. However, it is not the classic hosting because, unlike hosting, the provider provides the same application to all tenants. The model II shows a multi-tenancy platform via shared hardware (Elastic Virtualization). Each tenant has its own dedicated stack of technology, but the (virtual) machines are allocated from a pool of shared hardware. This model shares many characteristics of isolated tenancy because most of the stack is dedicated to a tenant. But it can be true multi-tenancy, as long as the underlying machine resources are shared in an elastic manner among the tenants. The model III shows a multi-tenancy platform via shared process. Multiple tenants' transactions are executed in the same instance of an application container (SaaS/cloud-enabled application platform, a SEAP). But each tenant is associated with a separate logical or physical instance of the DBMS. Execution environment is shared but data platform is not. Premise of this approach is that DBMS isolation would ensure the integrity of tenants' data. The model IV shows a multi-tenancy platform via Shared Everything. This is an enhanced variant of the SEAP-based multi-tenancy model. The application execution platform and the data platform are both shared by multiple tenants under the management control of SEAP. The model V shows a multi-tenancy platform via custom application design. Some SaaS-style applications have been designed using conventional programming languages over conventional third-party platforms, yet are offered as a multitenant service. Typically, such applications implement the principles of multitenant isolation as part of the application code itself.

One of the big advantages for SaaS startups is the opportunity to be economically efficient along many dimensions through multi-tenancy. A multi-tenant architecture, in which all users and applications share a single, common infrastructure and code-base is centrally maintained. Because SaaS vendor clients are all on the same infrastructure and code-base, vendors can innovate more quickly and save the valuable development time previously spent on maintaining numerous versions of outdated code.

Figure 2 shows an ideal multi-tenancy enablement framework[2] when the multi-tenancy platform model IV of architecture is applied above. This framework is mainly to serve the small business tenants who care typically about the costs without significant loss in utilizing the applications in general. Most of application logic oriented developers are multi-tenancy non-awareness and responsible for developing or customizing the content of the UIs, business processes and services. While some infrastructure oriented developers should ensure the application can run efficiently and reliably at the minimum costs. *Multi-tenancy enablement layer* is the core of the entire framework, which provides the clear separation between the usage of the applications and the supporting system resources. The fundamental contribution of the multi-tenancy enablement layer is to let the tenants enjoy the benefits of the native multi-tenancy while hiding the complexities of enabling the capability from both the tenants and the business logic developers.

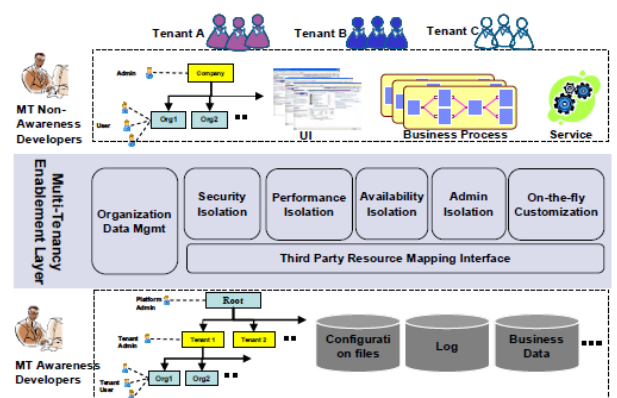


Figure 2. A Ideal Multi-tenancy Enablement Framework[2]

On the Figure 2, to provide the tenants with the single-instance multi-tenant service, the role of the *multi-tenancy enablement layer* is necessary. The five main service components of the layer are as following:

- Security Isolation: In multi-tenant system, potential security risks introduced by other tenants who share the same application instance and resources have to be considered.
- Performance Isolation: When a multi-tenant system is operated, the tenants are naturally concerned about how to prevent one tenant's behaviors from affecting the performance of other tenants.
- Availability Isolation: When many tenants share the same application instance, it is critical for the environment to detect any faults and stop the propagation of the faults at the earliest possible time.
- Administration Isolation: The administrative consoles should be isolated for individual tenants so that they can view and change the operational and business level data that are relevant to their own organizations.
- On-the-Fly Customization: It is necessary to customize the application to fit the needs of the tenant. For tenants served under this framework, the system downtime should be minimized.

III. SYSTEM DESIGN

We designed a multi-tenant supporting online application service system based on metadata model, shown as Figure 3, includes a SaaS platform server and the devices connected with it. The SaaS platform server follows the multi-tenancy platform model IV. in order to provide the advantages of SaaS platform, the opportunity to be economically efficient along many dimensions through multi-tenancy, in which all users and applications share a single, common infrastructure, with platform provider and tenants.

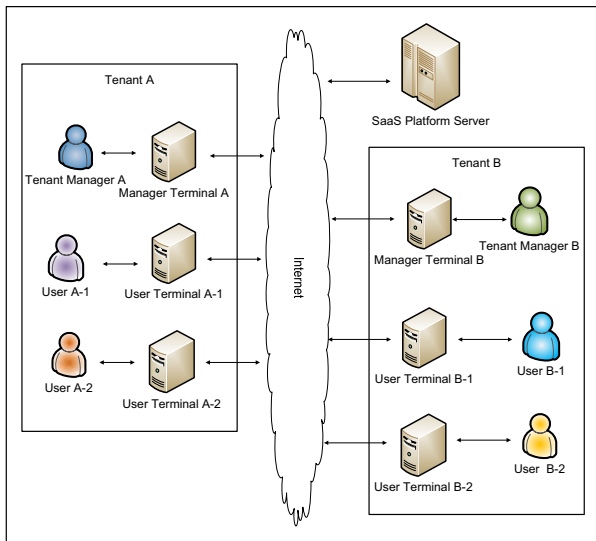


Figure 3. SaaS platform server and tenants

We present an online application service system which is applied code generation and metadata model technology in order to serve multiple tenants with a single instance of web based online application. The real service environment basically consists of a SaaS platform server and a plurality of tenants and their users, but in this paper, we present an multi-tenant architecture consists of a SaaS platform server and two tenants for example, shown as the Figure 3. On the service architecture, the two tenants use the same web based application served from the SaaS platform Server. Each tenant user or tenant manager terminal is connected with the SaaS platform server via internet or the other network service. The server has role of serving web based application which is customizable according to each tenant working environments. Each tenant consists of two tenant users and a tenant manager who customizes the application to fit the needs of their company.

Figure 4 shows the architecture of the SaaS platform operated on the server. The platform consists of the several service components. The *Presenter* composes user interfaces(UI) includes tenant manager UI serving customization roles and tenant user UI capable of using the application. For the tenant manager, also the presenter serves the roles of the *UI editor*, *Workflow editor* and the *DB schema editor*. By using the tools and through the *Configurator*, the customization or tuning is possible. When a tenant manager customizes an application, custom codes are generated by the *Code Generator* for only the tenant.

At Figure 5, the flow which is the process of controlling metadata, when a tenant manager tunes an application for the tenant, on the SaaS platform is presented. When the tenant manager customizes an application for the tenant, the *Presenter* requests the *Configurator* to create customized code. Then the *Configurator* creates the custom code which is the part of customized code. The *Metadata Encoder* creates metadata model and save the metadata to the *Metadata Database*.

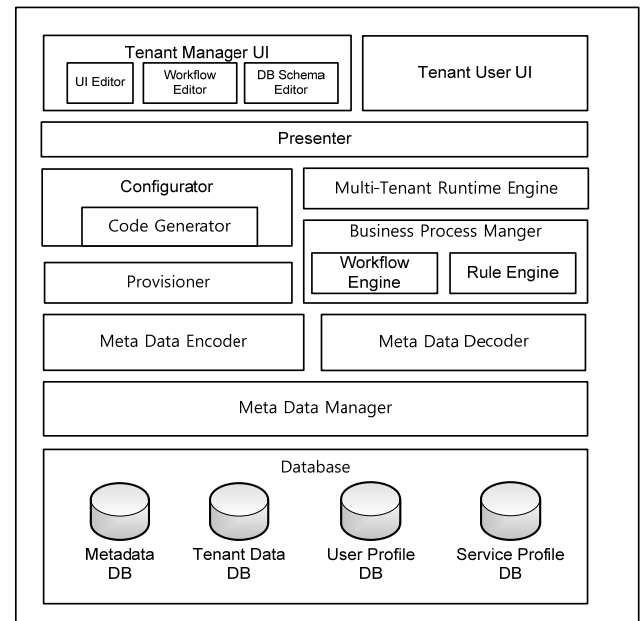


Figure 4. Architecture of SaaS platform

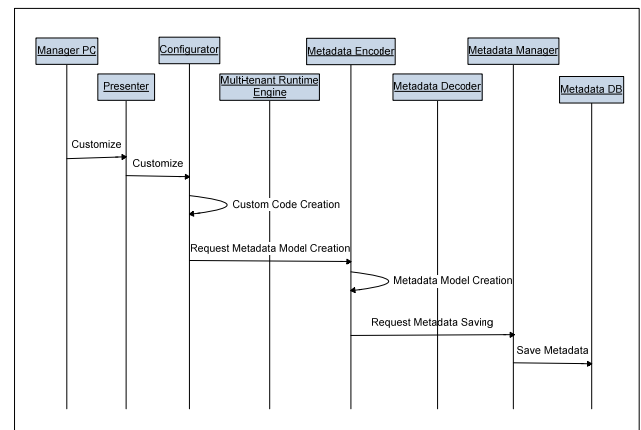


Figure 5. Stage of Metadata Model Creation

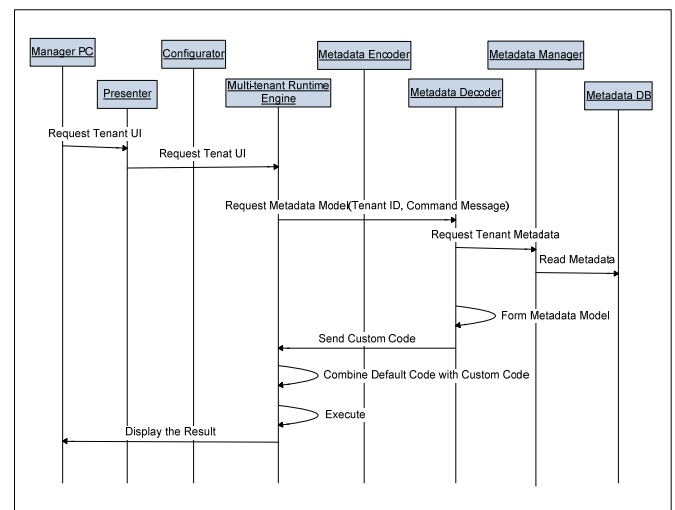


Figure 6. Stage of Application UI generation

At Figure 6, when the tenant manager or a tenant user make an event on the application UI, an command of execution of a workflow or edit database schema, the *Multi-tenant Runtime Engine* requests, with the tenant ID and a command message, the *Metadata Decoder* to get metadata model for the tenant. The *Metadata Decoder* gets the metadata according to the tenant ID and the message and creates metadata model.

The metadata model is delivered to the *Multi-tenant Runtime Engine* and executed after putting together with the application default code. The execution results are presented to the tenant manager or a tenant user as an application UI.

IV. CONCLUSIONS

We presented an online application service system which is applied code generation and metadata model technology in order to provide the advantages of SaaS platform model, the opportunity to be economically efficient along many dimensions through multi-tenancy, in which all users and applications share a single, common infrastructure, with platform provider and tenants. On the system, for customization, custom codes are created when a tenant manager configure an application. And the custom code and default application codes are put together to present the application according to the tenant. The multi-tenant

architecture, in which all users and applications share a single, common infrastructure and code-base is centrally maintained. Maintaining a single code-base would be simpler and cheaper for the SaaS vendor than maintaining each application of customers because the customers are all on the same infrastructure and code-base.

REFERENCES

- [1] Y. V. Natis, "Reference Architecture for Multitenancy: Enterprise Computing in the Cloud," Gartner, Dec. 3, 2008, pp.4-8.
- [2] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang and B. Gao, "Framework for Native Multi-Tenancy Application Development and Management", CEC-EEE 2007, 2007
- [3] P. K. Arya, V. Venkatesakumar and S. Palaniamami, "Configurability in SaaS for an Electronic Contract Management Application", Recent Advances in Networking, VLSI and Signal Processing, pp.210-216
- [4] Salesforce.com Corporation, <http://salesforce.com>, 2010
- [5] F. Chong and G. Carraro, "Architecture Strategies for Catching the Long Tail," Microsoft, Apr. 2006. <http://msdn.microsoft.com/en-us/library/aa479069.aspx>
- [6] Salesforce.com Corporation, "What is SaaS?", <http://www.salesforce.com/saas/>
- [7] "Software-plus-Services", Microsoft, 2009.
- [8] "Force.com: Create and Run any Application, On Demand," Salesforce.com
- [9] Software as a Service (SaaS) Resource Center, <http://www.deitel.com/ResourceCenters/Web20/SoftwareasaServiceSaaS/tabid/1659/Default.aspx>