2010 IEEE 6th World Congress on Services

A Transparent Approach of Enabling SaaS Multi-tenancy in the Cloud

*Hong Cai, IEEE Senior Member, Ning Wang, Ming Jun Zhou
IBM China Software Development Laboratory
*caihong@ieee.org

Abstract — **It has become more and more obvious that in Cloud Computing, applications are key drivers to make Cloud business a success. Multi-tenancy is a critical technology to allow one instance of application serving multiple customers at the same time to share Cloud resources and achieve high operational efficiency. There are different options of realizing multi-tenancy, in this paper we describe a transparent approach of making existing Web applications to support multi-tenancy and run in the public Cloud. Our approach includes intercepting the Web requests and deriving the tenant context, carrying the tenant context with a thread in the Web container, manipulating the isolation points (application artifacts that need to be isolated) in a Web application, and propagating tenant context to remote Cloud resources (such as database server, and message queue server) when necessary. With this approach, volumes of existing Web applications could quickly be provisioned through a public Cloud platform without rewriting the original source code. We have also implement a real system based on the common multi-tenancy model that separate the concerns of application developer, SaaS operator, tenant administrator, and tenant user. We finally integrate the SaaS multi-tenancy technique with Cloud platform services.**

Keywords-SaaS; Multi-tenancy; Cloud Computing; Public Cloud;

I. INTRODUCTION

In recent years, we have seen the evolution of research on Utility Computing, Grid Computing, virtualization technology, SaaS [1], next generation Web technologies, and Services Computing that have yield a new research area named Cloud Computing [2]. Cloud Computing promises to provide easy to use service oriented user experience to allow massive end users to use IT resources and IT applications as services without on-premise installing the IT infrastructures. Amazon EC2 [3] is a well-known and typical public Cloud offering.

A technical definition [4] of Cloud Computing is "*a computing capability that provides an abstraction between the computing resource and its underlying technical architecture (e.g., servers, storage, networks), enabling convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction.*" This definition states that Clouds have five essential characteristics: on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service.

This definition by default assumes the Cloud services as Infrastructure as a Service (IaaS). IaaS could be very successful in development and testing environment, because those public Cloud offering cheap and instant infrastructure services could ease the reach to IT resources. On the other side, through two years practices in the Cloud market, we have witnessed that customers in different geographies typically have different market needs of using Cloud services. For example, in the developing market there are strong needs of e-commerce, accounting and ERP applications for small and medium business (SMB), so these SMBs can save investment on IT infrastructure and spend more budget on their direct business operations such as marketing and businesses development. With this intent, SaaS is often their first resort. Today, some business applications and software have been developed to a stage that most of the functions are good enough to most of the customers. The ultimate goal of SaaS is to provide software to those segmentations of customers with flexible deployment (without on-premise installation) and pricing model (usually by monthly or annually subscription fee). However, SaaS has different maturity levels with varying capabilities and cost structures [5]. In its basic levels (Level 1/2), lower layer IT infrastructure resources are shared. In Level 3/4, multi-tenancy is the core feature with single instance of application serving multiple customers (tenants) to achieve the minimum operational cost, and maximize the revenue of the SaaS business.

A key concept in SaaS is *isolation point*. *Isolation points* are Web application artifacts that need to be isolated for different tenants. An example of *isolation point* is a visitor counter implemented with static field. Requests to different tenants need to trigger the counters to increase their counts separately. Details of *isolation point*s will be introduced in Section III. To serve multiple tenants, a SaaS application needs to have all its *isolation point*s identified and isolated at runtime. Multi-tenancy is the core technology of SaaS, and we need to understand there are different options to implement multi-tenancy to enable Web applications which have Web UI, business logic, and database. One option is taken by Salesforce.com [6] who provides Web based SaaS application development based on a template of the application (such as CRM application). The problem of this option is that the application developers have to be trained to learn the new programming model. The limit is that there's little fine granularity customization exposed such as adding, updating, or deleting a class, a method, or a field. The second option of realizing multi-tenancy is through designing multi-tenancy entry points at design time [7], so that different tenants may have different service components through different entry points. This approach will allow the developers to follow the original program model they are familiar. The limit of this option is that the designer and developer of the application need to design all the artifacts

978-0-7695-4129-7/10 $26.00 © 2010 IEEE
DOI 10.1109/SERVICES.2010.48

that need to be isolated before implementation. The third option targets for existing Web application providers [8-10]. In this approach, an existing Web application will be migrated to be multi-tenancy enabled. At the same time, application server runtime and database server runtime will be instrumented to route the requests to different tenants. This option could relieve the burden of application providers who have existing Web applications with massive customers using them. These application providers could use the SaaS multi-tenancy enabling tool and runtime to try their application in SaaS market quickly. In this way, they expect to serve more customers while still achieving SaaS operation efficiency. Our paper focuses on the third option of making multi-tenant applications. The ultimate goal of this option is to provide a transparent approach for Web application developers.

This paper is organized as follows. In Section II, we define the major problems we will face if we want to achieve the transparent SaaS migration approach. In Section III, we define the conceptual model of SaaS multi-tenancy enablement. In Section IV, we walk through the end to end method, tooling, migration activities, and roles of SaaS multi-tenancy transparent enablement system. In Section V, we give the related work. We close the paper in Section VI.

II. PROBLEM STATEMENTS

The first problem we want to solve in this paper is the foundation of SaaS multi-tenancy which is isolation and customization for Web applications. And we assume that these applications are deployed in distributed middleware components in a product system. In [8], we have seen methods of making application server to support multi-tenancy. But there are some limits there and are extended by this paper. We know that in large-scale Web application production environment, we usually use clusters of middleware such as a cluster of Web server, a cluster of application server, and database server (hard to make itself a cluster). We not only need to make application server to support multi-tenancy, but also need to propagate tenant context information to the distributed middleware components in the Cloud.

The second problem is how to provide separation of concerns when realizing SaaS multi-tenancy. In the whole life cycle of software engineering, there are multiple roles involved. The introduction of multi-tenancy may trigger additional issues. We need to separate the concerns of different roles so that they could focus on their own tasks without being bothered by the introduction of multi-tenancy aspect. A developer does not need to learn multi-tenancy but should focus on the application itself. The middleware provider would like to provide an add-on to the existing middleware (such as application server, database server, etc.) without maintaining two editions (one for traditional Web applications, one for multi-tenant Web applications). The SaaS/Cloud platform operator would like to minimize the effort of deploying and provisioning the (multi-tenant) application without additional deployment and administration efforts. The customers (tenants) of the multi-tenant application may wish to do some customization through self-service approach without interfere of SaaS/Cloud operator. What is not covered in [8] is how to integrate the life cycle of SaaS multi-tenancy work with the life cycle of Cloud enablement. SaaS multi-tenancy and Cloud virtualization technology are two complementary technologies. SaaS multi-tenancy targets for the finest granularity resource sharing with IT stacks (machine, OS, middleware, application), which is independent with the options of low-level machine resources (bare metal machine, or virtual machine (VM) [16]). On the contrary, Cloud virtualization technology is developed in an era when the capability of low end servers (such as Blade Server [17]) are becoming more and more powerful and can be easily virtualized to provide multiple VMs which could bring much higher flexibility to the infrastructure management, and much higher utilization of the servers (say from the normal utilization of servers in a data center of 10%, to 70%). SaaS multi-tenant applications are Web applications with certain workload characteristics, so it's helpful to combine the power of Cloud (virtualization) technology with SaaS multi-tenant technology, and combine the Cloud operation life cycle with SaaS multi-tenant life cycle.

We solve those problems in Section III –IV respectively. We should point out that SaaS multi-tenancy solution has dependency on the programming model. In this paper, we adopt the Java EE programming model but should be able to extend to other Web programming model as well.

III. THE CONCEPTURAL MODEL OF SAAS MULTI-TENANCY

A. *Topology of Modern Web Applications*

The production level topology of modern Web applications normally has the similar configuration as shown in Fig. 1. It contains a cluster of Web server, a cluster of Web application server, a database server (could be in master/slave mode), an LDAP [11] server, and sometimes a cluster of distributed cache.

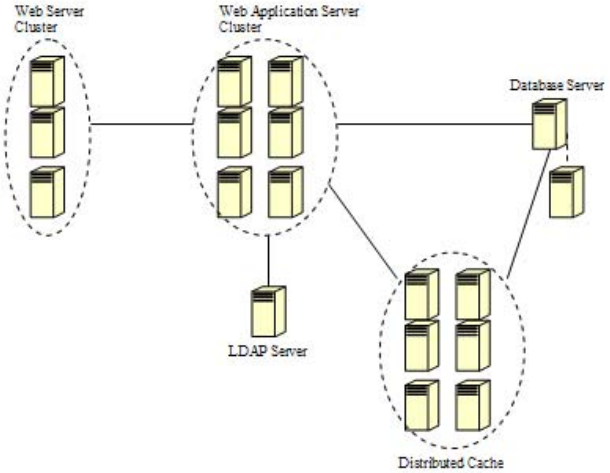

Figure 1. Topology of a production stage Web application deployment

The topology of a Web application deployment consists of

Web application servers and database servers. On a Web application server, we usually have a Servlet container, as well as an EJB container. A Web application is deployed on a Web application server. A database server is usually installed on another machine devoted to database processing, and serving the role of a remote resource. Beyond application server and database server, there could exist other remote servers such as messaging server, distributed cache server, etc. For example, a remote LDAP server may provide directory access service; a remote message queue server [12] may provide messaging service. From multi-tenancy perspective, a critical need of multi-tenancy enablement is intercepting tenant context information (such as tenant identifier) at the front end of a Web request. At the application server tier, the tenant content information will be carried by a thread and be used to route to tenant specific application server classes throughout the life cycle of the thread. The tenant context needs to be propagated to other remote resources (servers) for a Web application to complete its Web application processing.

B. *Isolation and Customization in SaaS Multi-tenancy Technology*

Isolation and customization are two major needs for SaaS applications to be multi-tenancy enabled. Within those two factors, isolation is the foundation of all enablement technologies, and customization provides tenant unique faces or behaviors.

1) Concept of Isolation and isolation points: An *isolation point* means a specific Web application artifact (class, method, or field) that has its tenant specific behavior or value. So it needs to be isolated for a specific tenant. The *isolation point*s in standard Java Web applications could be categorized in Fig. 2. The *isolation point*s could be identified at the application migration phase, and be stored in a metadata repository. Only based on isolation can SaaS applications provide customization capability in a consistent approach through application provisioning and configuration without being hard coded into the application.

	Type of isolation points	Where it is used
Type 1: Application level isolation	App Scope Objects	▪Global variables (including static field, global objects, cache, etc.)
		▪Literals (constant)
		▪Application cache
		▪Servlet Context
		▪Servlet Context Application configuration files
	EJB	▪SessionBean
		▪EntityBean
	Authentication & Authorization	▪JAAS w/ global unique login name (e.g. email)
		▪JAAS wo/ global unique login name
		▪Other DB/LDAP Based security
	Remote Service Call	▪HTTP/REST/SOAP
		▪JMS
		▪Socket
		▪RMI
	File	▪Standard java IO interface
		▪Logging
	Type of isolation points	Where it is used
Type 2: Resource level isolation	Database	▪Pure JDBC
		▪Common Data Access Framework (e.g. JPA)
		▪EntityBean
	LDAP	▪Basic LDAP
	Message	▪JMS
		▪MDB

Figure 2. A complete list of all (Java) Web application isolation points.

But we want to point out in this paper that the low level isolation points (such as global variables and literals) will be handled by our JVM level SaaS multi-tenancy enabling technique automatically. And only high level isolation points (such as file name and some UI artifacts) will be exposed and allow tenant administrator to make some simple configuration as illustrated in Section IV.

2) Customization: For example, a visitor counter on the home page of a Web application could be implemented using a tenant specific label plus the counted number using static field. Customization is handled during tenant onboarding phase. When a new tenant subscribe to this application, the tenant administrator could customize this counter's label through setting a tenant specific string.

C. *The Conceptual Model of SaaS Multi-tenancy in Web Application Programming Model*

As shown in Fig. 1, in production environment a Web application is often deployed in a distributed system consisting of many middleware components running on many physical machines or VMs in the Cloud environment.

In order to make those existing Web applications

We define a Web request as *q*, and the multi-tenant interceptor as *I*.

We define the tenant context for the M[th] tenant as $\mathrm{t(M)}$.

For each Web request, we derive the tenant, and bind the tenant context information to a thread *T* generated by the application server :

$$I(q) \rightarrow \vec{T}^{\{M,t(M)\}}.$$

For each multi-tenant Web application, we need to have a shared global map P to store tenant specific information (such as tenant specific field) on the application server. We define the partition for tenant M in the map as P(M).

We denote W(M) as some Web application business logic. We use $\Rightarrow$ to denote the transformation (routing) of an invocation from an original JVM level or Web container level API R_{jo} to a new multi-tenancy aware API R_j,

W(M) call $R_{jo} \Rightarrow$ W(M) call R_j * update P(M).

Sometime, W(M) needs to access a remote resource. It starts by calling the local resource client which has been replaced with a multi-tenant version on the application server,

W(M) call $C_{jo} \Rightarrow$ W(M) call C_j * update P(M).

At this time, the tenant context information should be propagated from the local resource client to the remote resource server. We denote the remote service server of type *i* with S_i. In the multi-tenancy scenario, $S_i(M)$ is used to denote the M[th] partition for tenant M in the remote service server (e.g., a DB server).

Figure 3. The conceptual model of SaaS multi-tenancy covering interceptor, tenant context, multi-tenant partitions, routing mechanism, and tenant propagation.

become multi-tenancy applications, we not only need to get the tenant context information, but also need to propagate the tenant context information to those distributed systems offering remote resources that the Web applications will use. Since there are many different commercial or open source implementation of the Web application programming model, we need a conceptual model to capture the most critical and common artifacts for enabling multi-tenancy. We propose a conceptual model of multi-tenancy in Fig. 3.

We could see in Fig.3 that the conceptual model of SaaS multi-tenancy includes multi-tenant interceptor, tenant context, (in memory) shared map to store tenant information, original application logic (using threads), JVM or Web container APIs, and remote servers. Multi-tenancy interceptor could be implemented with a Web filter and capture tenant information by knowing which tenant this Web request is for. Tenant context is an object to store tenant information. Tenant context could be bound with a thread and used by application business logic codes. The invocation of JVM and Web container APIs will be intercepted and added with some multi-tenancy specific pre or post operation. Tenant context information can also be propagated by a client (e.g., JDBC client) of remote service to the server (database server) of that service.

IV. The SaaS Multi-tenancy Tooling and Life Cycle Manager to Separate Concerns of Different Roles

A. Roles Involved in SaaS Multi-tenancy Enablement

1) Application Devleoper: The Web application developers are responsible for developing the UI, business logic, and database of the application. It's very hard for them to learning in-depth knowledge of SaaS multi-tenancy. Keeping the original user experience of developing the Web applications is very important for the application developer team to help migrate the single instant application to a multi-tenant application. On the other hand, the developer team is familiar with the structure of the application package, so they are responsible for identifying the *isolation point*s and export those *isolation point*s to metadata repository of the application. This is the linkage between SaaS multi-tenant application and SaaS platform. The application developer team does not need to understand how the SaaS platform operate.

2) Cloud/SaaS Operator: In the traditional application development life cycle, application administrator will be responsible for deploying applications. In the world of Cloud and SaaS, a Cloud and SaaS operator will be responsible for deploying applications to the SaaS platform, registering the application as an offering so that customers could subscribe to it, and defining the charging policy (matching different payment with different SLA (service level agreement)). SaaS operator will be also be responsible for approving the request from a tenant administrator (described below). SaaS application provisioning is different with traditional application deployment in that a multi-tenant application cannot be used after deployment. Only after a tenant administrator subscribes to the application, and make tenant specific configuration and customization (such as uploading the logo image, configure the isolation points with tenant specific data) can the multi-tenant application be used by that tenant. Beyond these activities, the SaaS operator will also be responsible for activities such as defining different level of SLA, defining the policy of allocating remote resource pools (at least database resources) for different SLA. Through this approach, the developers do not need to configure multi-tenant application resources statically at design and development time. Rather, the multi-tenant resources will be exposed as services and bound to an application when a tenant begins to use the application.

3) Tenant Administrator: First, a tenant administrator (TA) is responsible for subscribing to the SaaS multi-tenant application by agreeing to the terms and conditions of the service. Second, the TA is responsible for making tenant specific simple and high level configuration and customization. At this stage, the TA may also need to upload tenant specific data to the SaaS platform. Third, the TA is responsible for creating account for the end users of the tenant. At the end of a TA's operation, a URL to the entry point of the application will be provided. The TA could then notify its end user of the entry URL of the application.

4) Tenant User: As an end user of a multi-tenant Web application, the user will be provided with an initial pair of username and password after its account has been created by the TA. The tenant user could then access the entry URL of the multi-tenant application. The UI of the application will be tenant specific just like the tenant user is accessing an application that is unique to this tenant.

B. Tooling for the Application Development Team

We build the SaaS multi-tenancy tooling as a simple Eclipse based plug-in that only adds a new "*isolation point*" view to the traditional Eclipse IDE (Integrated Development Environment). A snapshot of the SaaS multi-tenancy tooling is shown in Fig. 4.

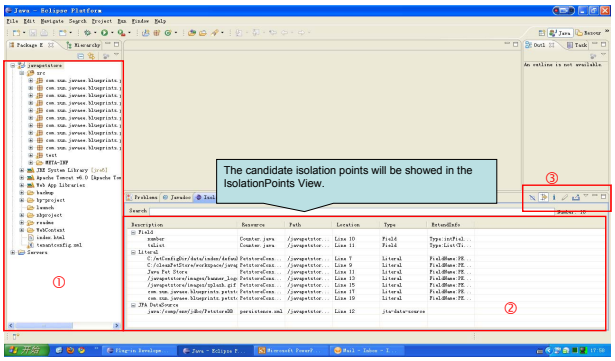

Figure 4. The snapshot of SaaS multi-tenancy migration tool as an Eclipse plug-in.

Part 1 (marked ①) of the tooling canvas is the traditional Java Web project explorer. Part 2 and 3 (marked ②and ③ respectively) make up the "*isolation point*" view. Part 2 lists all the candidate *isolation point*s for the application developer. Part 3 consists of some useful buttons (such as "confirm" *isolation point*s, "search" *isolation point*s within

the results by key words, and "export" all confirmed *isolation point*s to a metadata storage. This provides a heuristic approach to narrow down the isolation points.

C. *The End to End Multi-tenant Application Provisioning Through Cloud Platform as a Service*

This paper only focuses on SaaS multi-tenancy and how it could be supported by Cloud Platform as a Service (PaaS). The details of PaaS are beyond the scope of this paper. Throughout the paper, we use DayTrader (a sample Web application in IBM WebSphere Application Server) as the example application.

1) Create the Virtual Images for the Multi-tenant Application Server Runtime and database Server Runtime: It's quite time-consuming to set up a multi-tenancy runtime from scratch. For the application server, it means a serials of boring actions including installation of original application server, installation of multi-tenancy add-on packge, configuration of the application server to connect to database server, etc. For the database server, the actions include installation of original database server, initialization of the multi-tenancy system database, etc. To streamline the process, we could create virtual images for the multi-tenancy enabled application server and database server. Any virtualization technology could be supported, such as VMWare [13], Xen [14], and KVM [15].

2) Create the Application Model: The purpose of application modeling is to provide an intuitive approach to define a common model for a Web application running on a Cloud topology to avoid the complexity of manually installing and configuring the nodes depicted in Fig. 1. A example of the application model with simplified topology (one application and one database server) is shown in Fig. 5.

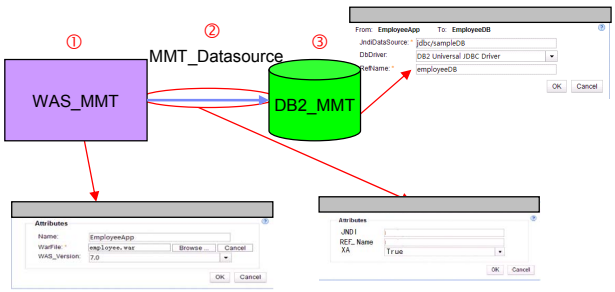

Figure 5. A simple application model.

The multi-tenant application model in Fig. 5 consists of an application server node (AppNode), a database server node (DBNode), and a link (representing constraint) from AppNode to DBNode. Each node or link has a corresponding property view under it that could be used to configure the parameters for that artifact.

For example, the parameters for the AppNode include the name and version of the AppNode, and names of the applications to be deployed at this time. The parameters for the DBNode include JDBC driver version, etc. The parameters for the link include datasource, etc.

3) Transform the Application Model to Topology Model: The topolgy model is the bridge between an abstract application model and Cloud infrastructure services which are based on VMs. An example of the topology model file is shown in Fig. 6. A topology model is an aggregation of definition of all nodes that make up the cluster built with virtual images. For each node, there is a definition of image, and a definition of parts. The"image" section contains the image ID and URL of the image, where the PaaS services know where to download an image for that node. The "parts" section contains aggregation of a series of parts. In our multi-tenancy scenario, we should have a separated part for original middleware component (application server, and database server) and an additional part for the multi-tenancy components on each server. This is to separate the PaaS management for original SaaS middleware and multi-tenancy add-ons. Once a virtual image is created, it can then be captured as a combined virtual image (snapshot) to be quickly provisioned in future use.

```
{
 "topology_vms" : [
  {
   "id"  : "vm_App",
   "image" : {
    "location" : "http://imagestore/.../images/AppImage/",
    "type"    : "m1.small",
 "parts" : [
       {"part" : "http:// imagestore /.../AppNode.zip"},
       {"part" : "http:// imagestore /.../App2DBLink.zip"},
       {"part" : "http:// imagestore /.../AppMulti-tenancy.zip"}]},
  }

  {
   "id"  : "vm_DB",
   "image" : {
    "location" : "http://imagestore/.../images/DBImage/",
    "type"    : "m1.small",
 "parts" : [
       {"part" : "http:// imagestore /.../DBNode.zip"},
       {"part" : "http:// imagestore /.../DBMulti-tenancy.zip"}]},
  }
```

Figure 6. Example of a topology model for the multi-tenancy enabled in a Cloud.

4) Instantiate the Virtual Images on the Cloud Platform through PaaS Services: Figure 7 shows the high-level activities of PaaS services that accept the topolgy model, invoke the IaaS layer APIs and instantilize the virtual image, and run extra activation code, until fully set up the complete cluster consisting of the nodes defined in the topology file. The detailed architecture and process of the PaaS services are beyond the scope of this paper.

The first step in the PaaS management process is to parse the topology as described in Fig. 6 based on coming request of creating a multi-tenant application topology in the Cloud. After the topology file is parsed, the image ID and image location, as well as parts information will be derived. The files described in the parts will be retrieved from storage and will be installed on a base image described in the base image section.

After that, the PaaS services will call IaaS APIs to start the image. IaaS services will be responsible for allocating

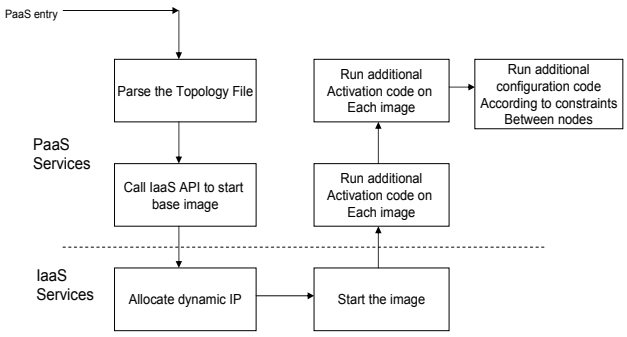

Figure 7. High level flow of instantiate the virtual images on the Cloud platform through PaaS service, it will further invoke IaaS API to get IP and start VMs.

elastic IP addresses, assigning the IP to the virtual image and starting the virtual image into a VM. VMs will be configured according to the "type" attribute defined in the topology file.

Now, the base image in the new VM has been started, some additional activation codes could be executed as system services of the base OS (customized for the PaaS).

After the VMs in the cluster have been started, some additional codes will be executed to fulfill the constraints between nodes of the cluster. For example, an application server node needs to configure the data source to use an existing database server node. Because of this, the application server node needs to have the IP address of the database server node. This will be the last step of activating the middleware nodes in the cluster.

At last, the application configuration code could be executed. Some initial application data could be generated and saved in the system table of the application.

D. Use Life Cycle Manager Application to Make SaaS Multi-tenant Applications as Offerings and Make the Tenants Onboarding

1) (Subscriber) Making the New Multi-tenant Application into an Offering: An offering is different with an application in that an offering is based on an application (functional components) and contain other artifacts such as SLA and matching price (non-functional components). The 3-tuple {application, SLA, price} makes up a SaaS billing policy. A snapshot of massive multi-tenancy provisioning application (administration console) is shown in Fig. 8. An offering management application may cover many functions such as creating a new offering, updating an offering, etc. When creating a new offering, some compulsory information must be filled or selected such as the name of the offering, types of SLA this offering supports, whether this offering needs to be activated immediately, entry point of provisioning service (to be used to make the tenants onboarding). In the example in Fig. 8, we can select how many different types of SLAs this offering can support. The types of SLA may vary from the most cheap and low resource consuming "economic" type (shared table approach), to "intermediate" type (separate schema), to "advanced" type (separate database), till the most delegated "deluxe" type (separate database instance).

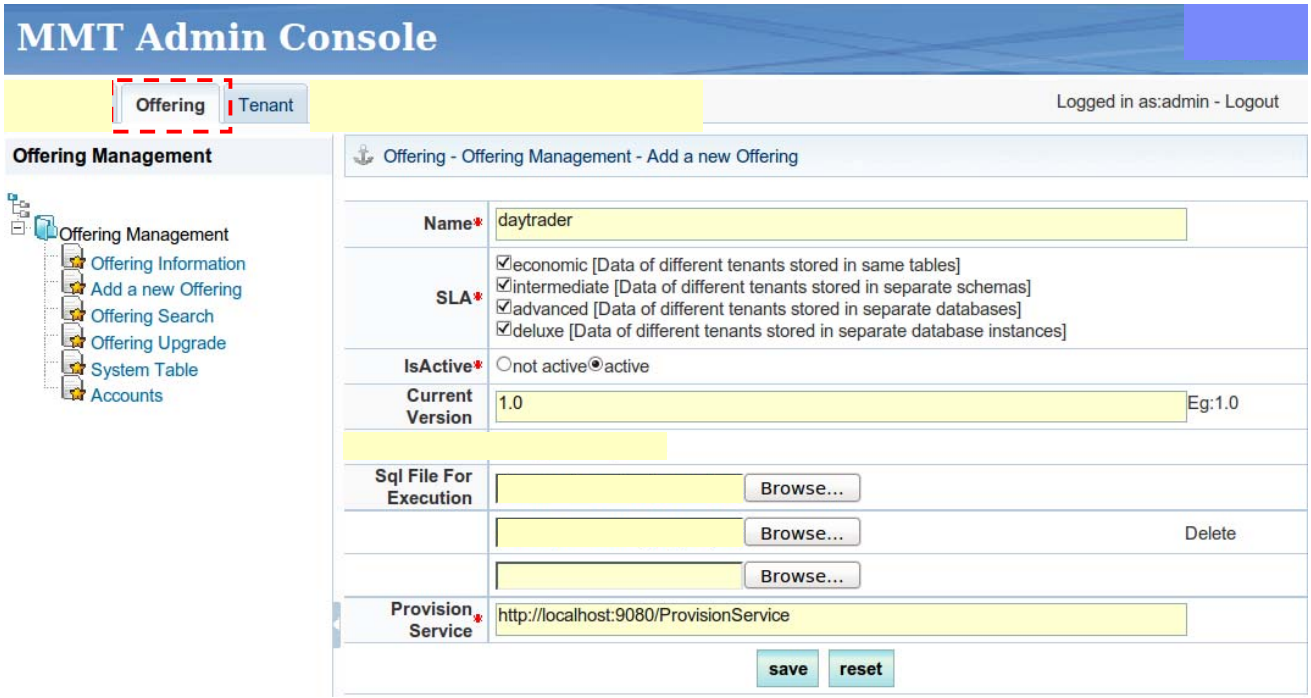

Figure 8. Making the new SaaS multi-tenant application a SaaS offering.

2) (Tenant Administrator) Subscribing to a Multi-tenant Offering: As shown in Fig. 9, a tenant administrator could subscribe to a new offering through self-service approach or

through a client representative. If the account of the tenant has already been created, then the tenant administrator could log into the system and select the name of the new offering, then select the type of SLA the tenant wants to use. At the backend, the SaaS multi-tenancy operation service could automatically provisioning necessary resources and create partitions for the new tenants of the offering.

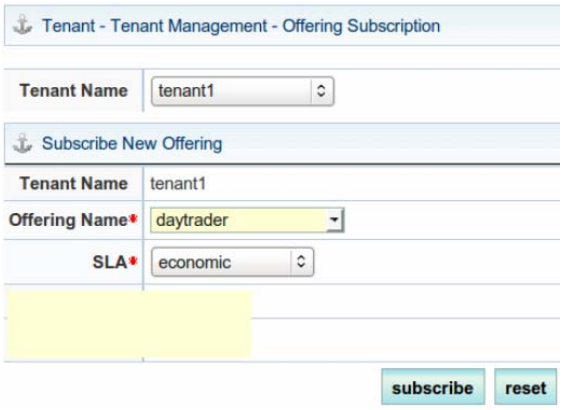

Figure 9. Select a SLA for the new tenant.

3) (Tenant Administrator) Customizing the Multi-tenant Application for the Tenant: As said, the difference between a multi-tenant application with a traditional Web application comes from the introduction of "tenant". Before a tenant administrator makes tenant specific customization, a tenant cannot use the application. Fig. 10 shows the snapshot of customization UI for a tenant administrator.

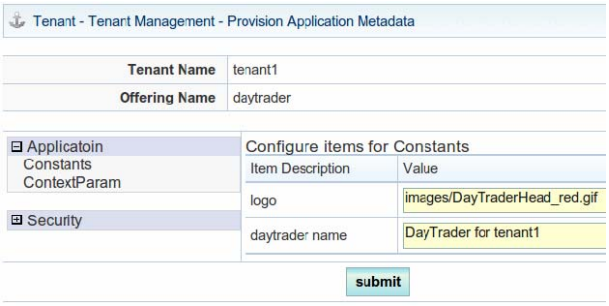

Figure 10. The configuration UI for a tenant administrator.

The contents shown in the UI in Fig. 10 vary from applications to applications. The contents will be automatically generated based on the metadata of the multi-tenant application which is derived from the application transformation phase (described in subsection B of Section IV). Not all *isolation point*s have customization points. Only high level *isolation point*s that have tenant specific initial or permanent values need to be exposed at this page for TA to customize. Examples of the customization points include logo (image or literal) of a tenant, labels (literals), etc. A TA will not be bothered by low level isolation points such as static fields, literals, etc.

4) (SaaS Operator) Monitoring the Use of the Multi-tenant Applications from Tenants: It's invevitable that some of the tenants have more aggressive use of the SaaS resources which is unfair for other tenants sharing the same resources (such as application server or database server). There are different ways to deal with this such as setting the threshold for maximum Web request within a unit time for a tenant. A multi-tenant monitoring and control mechanism could be set up to support daily operation for the massive tenants. A snapshot of the multi-tenant monitoring UI is shown in Fig. 11. It gives a clear picture of how the multiple tenants shared the resources (e.g., VM resource) in a Cloud platform.

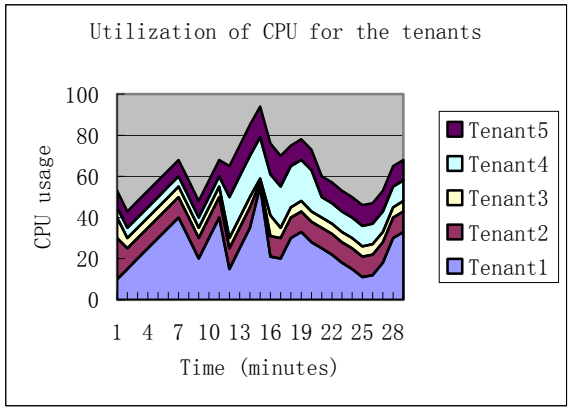

Figure 11. An example of monitoring multi-tenant applications.

V. Related Work

The related work mentioned below is related to the transparent approach of SaaS multi-tenancy enablement. As said in Section I, we understand there are at least two other approaches. The first approach is about totally changing the programming model. The second approach is about design for multi-tenancy at application design time. So these two approaches were not comparable to the work in this paper.

Database multi-tenancy maturity models and implementation methods were introduced in [5]. But [5] didn't cover the end to end life cycle management of a Web application, specifically the enabling of multi-tenancy on the application server.

The study of comprehensive isolation and customization of SaaS multi-tenancy were introduced in [8-9]. But they have some limits such as incomplete transparent support for migrating static fields in a Web application.

A complete end to end transparent SaaS multi-tenancy migration method was introduced in [10]. But it didn't cover the life cycle management of SaaS multi-tenant application. We know that from "X (something) as a Service" perspective, application provisioning and tenant on boarding are two important stage in the whole life cycle of SaaS operation, so [10] missed this part.

This paper includes SaaS multi-tenancy enablement work in the whole life cycle of Cloud platform management which takes the latest progress of Cloud virtualization technologies.

VI. Conclusions

The contributions from this paper are multifold.

First, this paper describes the core SaaS multi-tenancy models consisting of tenant interceptor, tenant context, tenant map, tenant propagation, remote resources (such as database server, LDAP server, message queue server) in a distributed multi-tier Web system.

Second, this paper also introduces the end to end process of making an existing Web application to be multi-tenancy enabled, and separating concerns of different roles involved. So in the end to end process, the application developer still focus on the application itself, and could use the SaaS multi-tenancy enablement tool to migrate the application and derive the multi-tenant metadata of the application. The SaaS operator could then provision the application and make it an offering. A tenant administrator could use the SaaS multi-tenancy life cycle management application to make simple application configurations to fit the needs of that tenant. The end users of the tenant use the application and have the feeling that they are using an application specifically deployed for the tenant.

Third, this paper puts the SaaS multi-tenancy operation in the complete life cycle of Cloud platform services. It describes the method of creating multi-tenancy enabled virtual images, creating the application model, transforming the application model into topology model, instantiating the virtual images, then activating the image and making additional installation and configuration including the multi-tenancy related actions.

REFERENCES

[1] http://en.wikipedia.org/wiki/Software_as_a_service.

[2] http://en.wikipedia.org/wiki/Cloud_computing.

[3] http://aws.amazon.com/ec2/.

[4] Cloud Computing Definition, National Insitute of Standards and Technology, Version 15.

[5] "Architecture strategies for catching the long tail". April 2006. http://msdn2.microsoft.com/en-us/library/aa479069.aspx. Retrieved 2010-03-14.

[6] http://www.salesforce.com/.

[7] Carl Osipov, German Goldszmidt, Mary Taylor, Indrajit Poddar, "Develop and Deploy Multi-Tenant Web-delivered Solutions using IBM middleware", http://www.ibm.com/developerworks/webservices/library/ws-multitenantpart2/index.html.

[8] W. Sun, X. Zhang, C. J. Guo, P. Sun, and H. Su, "Software as a Service: Configuration and Customization Perspectives", presented at Congress on Services Part II, 2008. SERVICES-2. IEEE, 2008.

[9] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang, and B. Gao, "A Framework for Native Multi-Tenancy Application Development and Management", presented at E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007. CEC/EEE 2007. The 9th IEEE International Conference on, 2007.

[10] H. Cai, K. Zhang, M. J. Zhou, J. J. Cai, and X. S. Mao, "An End-to-End Methodology and Toolkit for Fine Granularity SaaS-ization", presented at Cloud Computing, 2009. CLOUD '09. IEEE International Conference on, 2009.

[11] http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol

[12] http://en.wikipedia.org/wiki/Message_queue

[13] http://www/vmware.com/

[14] http://www.xen.org/

[15] http://www.linux-kvm.org/

[16] http://en.wikipedia.org/wiki/Virtual_machine

[17] http://en.wikipedia.org/wiki/Blade_server