ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/299874556

A new framework for customizing ERP systems in a multi tenant SaaS environment

Conference Paper · March 2015

DOI: 10.1109/WSWAN.2015.7209089

CITATIONS
5

READS
771

2 authors, including:

Djamal Ziani
Al-Yamamah University
13 PUBLICATIONS **57** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

ERP Cloud Security View project

All content following this page was uploaded by Djamal Ziani on 29 April 2017.

The user has requested enhancement of the downloaded file.

A New Framework for Customizing ERP Systems in a Multi Tenant SaaS Environment

Djamal Ziani
Information Systems Department
King Saud University
Riyadh, Saudi Arabia
dziani@ksu.edu.sa

Asma AlShehri
Information Systems Department
King Saud University
Riyadh, Saudi Arabia
ashehri44@yahoo.com

ABSTRACT

Customizing cloud ERP systems is considered one of the biggest issues in the multi tenant environment. This paper aims to propose a new framework, allowing tenants in the cloud ERP to change the ERP's source codes in order to meet their requirements. We inspired by SAP systems to get better understanding of the structure of ERP systems, and to propose a web based ERP application. Afterwards, we illustrated how we improved the proposed web based ERP application to support the customization process in multi tenant environment. The proposed framework is based on a strong architecture that manages and controls the customizing of the tenants to tailor the cloud ERP system without affecting each other.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management –*software configuration management*. D.2.13 [**Software Engineering**]: Reusable Software –*domain engineering*, J.1 [**Administrative Data Processing**]: Business. K.6.3 [**Management Of Computing And Information Systems**]: Software Management –*software development, software process*.

General Terms

Design.

Keywords

ERP, Cloud, Customization, Multi-tenant, SaaS, SAP.

1 INTRODUCTION

Enterprise Resource Planning (ERP) Systems coordinate all resources and integrate information across an entire enterprise to ease the flow of information among the different processes within an enterprise, and facilitate the communication and information sharing across organizational units and geographical locations. Implementing ERP system in the enterprise achieves tangible and intangible benefits; it enables an enterprise to function as a single unit rather than as several separate units and work processes. ERP System and its related database can support number of different hardware, network configurations, and interfaces [1]. A well-constructed and implemented ERP can increase productivity and provide information reliability, data redundancy prevention, cost lowering, and scalability and a global outreach improvement.

Modern approaches involving rising technologies permit for a ground up reconsideration of the infrastructure of ERP systems. The Cloud Computing rising technology was leading the way in redefining ERP Systems [1]. Cloud computing is potentially one of the major revolution in the computing history. It has changed the way of providing IT services. The IT cost, which was presented a barrier for the organizations to deploy many cutting-edge IT services, can be reduced using cloud computing technology [2].

Now days, cloud services play an important role in ERP systems. Deploying ERP in the cloud as a service (SaaS) enables the consumers or the organizations to conduct their businesses virtually anywhere, and do not tied to the organization's internal network or data center [3]. The organizations can also concentrate on their business rather than IT issues. Furthermore, Cloud ERP can lower the total cost of ownership, especially start-up cost, there is no need to purchase expensive servers, networking and software licenses or to hire a large number of IT staff, in addition to cut the cost of operating all its equipment. Therefore, the savings can be 30% to 50% of TCO compared to on-premise ERP [4].

There are number of factors that should be considered in deciding whether and how to move to cloud ERP systems such as industry type, company size, solution complexity, security needs, and other organizational issues.

Cloud-based ERP provides a number of benefits especially for Small and Medium Enterprises SMEs. [5] Analyzed Cloud ERP benefits based on 17 papers published after 2006. The main benefits were; decreases the operating costs such as maintenance and upgrades, speeds up the implementation and reduces the time of providing new products, and improved accessibility, mobility, usability and scalability. In addition to focusing on business competitions, using advanced technology, facilitated integration with cloud services, improved system availability and allowing disaster recovery.

Many SMEs are willing to consider cloud ERP for the above-mentioned benefits that are seem obvious in this market segment, especially cost saving. One significant issue that should be taken into consideration when the company intends to subscribe on cloud ERP services is the standard processes that make the companies almost identical and remove the possible competitive advantages. In this case, the company may adjust the best practices of the ERP system depending on the processes really

executed or conversely [6]. Thus, the ERP vendors provide customization options, which allow the company to write a new code to add new functionality or to change the way the ERP runs. So, they can tight the gap among the company's unique processes and ERP best practices.

Customizing ERP in SaaS environment poses new challenges to cloud ERP vendors. With Multi-tenancy, whish is one of SaaS major technologies; where a number of companies (tenants) are served by the same application by sharing the same software and hardware infrastructure, the customizing done by one tenant may affect all other tenants, or may need complex and special development [6].

In this paper we proposed a framework to customize cloud ERP systems in multi tenancy environment. This framework will help the tenants to modify the code and customize their own services without affecting other tenants. Data isolation also will be considered in our framework.

2 RELATED WORKS

Several authors were proposed models or approaches to deal with the customization issue in SaaS multi tenant environment:

The authors in [7] have explained the difference between software configuration and customization, and the reasons for the requirement differences, such as customer behavior differences, which make the SaaS clients need the software tailoring (configuration or customization). Then, they proposed a competency model and framework to support the SaaS vendors in customization issues. The model has classified the SaaS configuration in 5 levels of competency, from level offering highly standardized software to the one that provides programing tools enable developing a new application.

The paper cited the different isolation levels of sharing database in multi-tenancy, with different levels of customization and security. The high level of isolation among different tenants leads to easier customization, but it may lead to more expensive hardware and maintenance. However, this paper gives only the big lines to follow in the customizing, the technical detail is missing.

[8] Suggested three approaches for sharing data in multi-tenancy systems with different levels of isolation. Shared Machine, in which each tenant can access its own database, and multiple tenants share the same machine. This approach presents a high level of isolation, but low level of resource pooling. Shared Processes, in which each tenant can access its own tables and multiple tenants share the same database process. This approach presents medium level of isolation, better level of resource pooling than shared machine approach. Shared Tables, in which data from many tenants are stored in the same tables. This approach presents low level of isolation, but it is the best resource pooling approach.

[9] Proposed domain engineering based business rule templates to express business logic, in which each tenant is equipped with a business object table (BOT). When tenants define business logic by choosing and completing predesigned templates according to their business needs through the visual rule definer, business objects relevant to business logic will be added to the BOT. When the software requires calling the rule engine, it will first inquire the tenant's BOT, and the system will request the rule engine only if it found a relevant business object in the tenant's BOT.

Other approach of customizing a SaaS application with multi tenancy was proposed by [10]. The author of this paper proposed a cooperative construction model based on multi level abstraction of business logic to create a new SaaS application. This proposed approach enables the tenant to be created from various components and through various constructing routes with the collaboration of various partners in a SaaS system. Where Service providers can sign up and upload a service component, and platform operator assembles the component for each tenant according to tenant's demand. Each tenant has assigned metadata folders to save its metadata files.

[11] Suggested a customization relationship model for supporting the multi granularity relationship in the multi tenant environment of SaaS. They classified the customized objects into data, service, process and user interface levels and addressed the relationship among the objects of each level and among the different levels. According to the relationships among the customized objects, they suggest four granularity levels, which are: parameter, object, cooperation and coding granularity.

Several authors have proposed using Orthogonal Variability Model (OVM) to support the customization in multi tenant environment. [12][13] Suggested a customization models based on OVM. Where a variation point is a component that can be replaced by other sub workflow templates (variants). [14] Proposed an MDA-based of SOA model for customizing the software in SaaS environment. They describe the design and implementation of a service template markup language (STML) and its incorporated development tools such as Service Customization and Integration Tools, the Service Execution Platform, and the Service Templates Repository, which present a thorough solution for customizing the software in SaaS environment.

[15] Presented an approach that helps the SaaS application developers to develop efficient customization tools for their applications, their approach illustrates how to use BPEL and generate customized processes out of templates and variability descriptors.

3 WEB BASED ERP APPLICATION

Typical ERP systems use three-tier client/server architecture by which hundreds of ERP clients can access these systems through a local area network (LAN)[16].

In this section, we will present a suggested web-based ERP application; inspired by SAP CRM system (figure 1), and in the next section we will explain how we improved this application to support many tenants and how they can customize it.

Our suggested ERP application mainly divided into two-part ERP front-end and ERP back-end, where ERP users can interact with the front-end ERP and the ERP front-end sends user requests to the ERP back-end.

The ERP front-end and ERP back-end are organized as follows:

3.1 ERP Front End

The ERP front-end is based on three-tier architecture. The tiers and their description are listed below:

3.1.1 Presentation Layer

The aim from this layer is to support the users with the user interface and enable the navigation within the application.

In our framework we supposed the use of Model-View-Controller (MVC) whish is "the name of a methodology or design pattern for successfully and efficiently relating the user interface to underlying data models."[17]. The main components of MVC pattern and their collaboration are shown in figure 2.

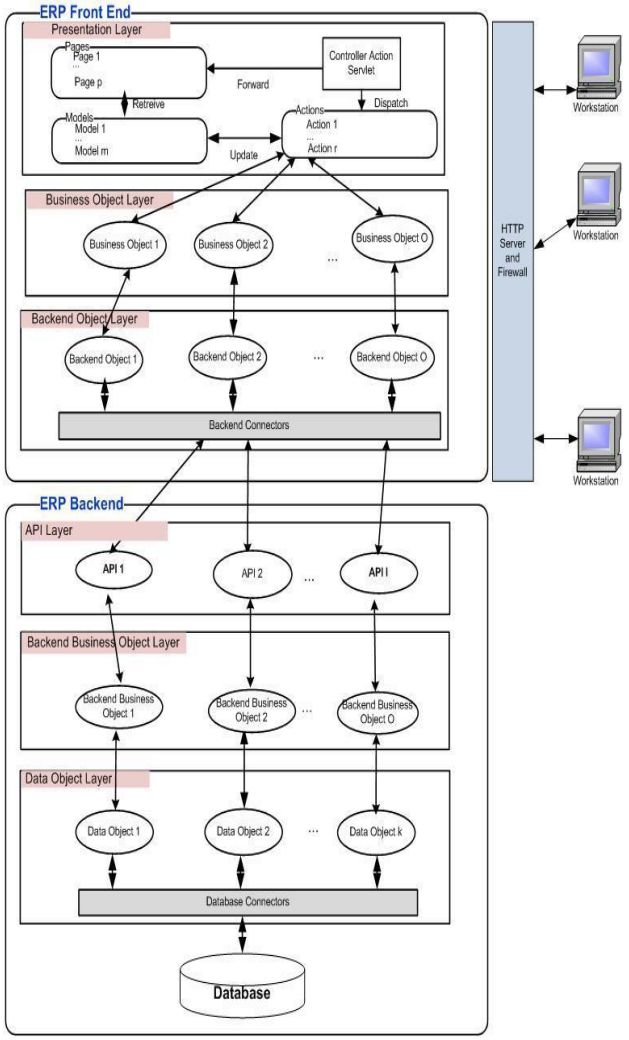

Figure 1. a Suggested Web Based ERP Application.

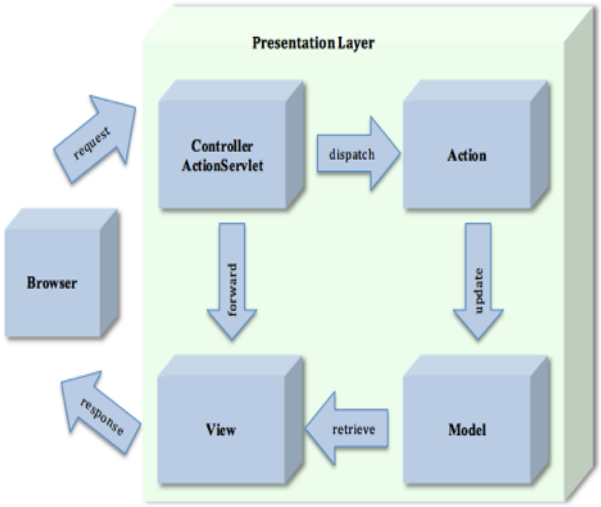

Figure 2. MVC Components.

Where:

- All the clients' HTTP requests are passing through the controller Servlet.
- Controller Servlet works as a dispatcher transmits all the received HTTP requests to the relevant Actions.
- Action works as a mediator between the HTTP world and Business Objects.
- Based on the returned data from the model, the controller servlet forwards the calculated data to view it and response to the users' requests after mixing the static HTML content with the dynamic content.

3.1.2 Business Object Layer

Business Object layer contains all the Business Objects (BOs) that interact with the presentation layer. These Business Objects provide part of the business functionality, and they represent the backend-independent part of the business logic. BOs are not aware of the backend specifications and they do not interact with the backend system directly, however, they do so through Backend objects in the Backend Object layer.

3.1.3 Backend Object Layer

This Layer enables the Business Objects to communicate with the back-end ERP, where BOs are not aware of the backend specifications. For each BO, there is a matched Back-end Object (BE) that provides an interface to the BO to access the back-end system. BE objects can communicate with the Back-end ERP using backend-specific connectors.

3.2 ERP Back End

The ERP Back-end is based also on three-tier architecture. The tiers and their description are listed below:

3.2.1 API Layer

Application Programming Interface (API) Layer functions as an interface that enables the ERP front-end to communicate with the ERP back-end. This layer contains a set of API functions; each one is responsible to answer to a backend object. The ERP users are not aware of this interface and how it works, it only used to pass the data between the ERP back-end and front-end.

3.2.2 Backend Business Object Layer

Back-end Business Object layer contains all the Back-end Business Objects (BEBOs), these BEBOs encapsulate the business logic. Each BEBO implements part of the business functionality. BEBOs may interact with each other to implement business process such as purchase order, creating invoice and purchase return. Back-end Business Object layer is responsible for interacting with Data Object Layer to retrieve, modify and access the data and pass it to API Layer.

3.2.3 Data Object Layer

This layer functions as an interface that allows retrieving and storing business objects data from/to the databases. The data object layer uses database connectors to send or receive data to/form databases.

3.2.4 Database

The Database contains all ERP business data.

4 PROPOSED FRAMEWORK

In this section, we will illustrate how we can improve the previously ERP application, described in the previous section, to enable each tenant that is using ERP application to customize the application in order to meet its own requirement without affecting the other tenants' functions.

4.1 Main Adding in the application architecture:

In order to have a secure, consistent, and an easy use, the new framework is based on three major adding:

4.1.1 Multi-Tenancy Application Manager

The Multi-Tenancy Application Manager is a very important component in the new framework; it has a responsibility for securing the access to the tenants' applications, building the tenants' applications, and guiding the tenants' customizations. It is composed by three components as shown in figure 3:

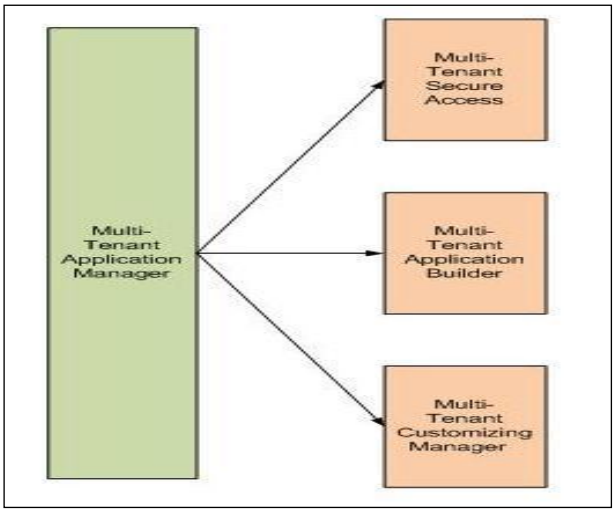

Figure 3. Multi-Tenancy Application Manager

- *Multi-Tenant Secure Access*: is responsible for filtering the access to the tenant's application. Each tenant application is accessed through the tenant's portal. Each user should log with a user password, and the portal will send a tenant ID in the background to the Multi-Tenant Secure Access. Using these three information (user, password, and tenant ID), the Multi-Tenant Secure Access will access the right tenant database (by using the tenant ID, it will call the database dispatcher and connector component), check the user in the tenant database, and allow or reject the access to the tenant application.
- *Multi-Tenant Application Builder:* The tenant application is made by the default provided application objects and by the application objects, which have been customized by the tenants. Using the customizing files the Multi-Tenant Application Builder will build in a correct and secure way the tenant application each time the tenant wants to use the new application version.
- *Multi-Tenant Customizing Manager*: Is a sort of smart wizard, responsible to guide the tenant in the customizing task: creating and saving objects to customize, updating customization files, show the dependent objects to customize.

4.1.2 Customization files and Layer Manager

In order to avoid hard coding the object names in the application code, since tenants can use different objects during the execution of application, we created in most layers a layer manager which is responsible to read the customization files, that contains the name and path of the objects, and load on runtime the right objects.

For example, the customization file of the business object layer has the following format:

```
<businessObjects>
   <businessObject name="LoginBO" default="com.erpCompany.LoginBO.java" />
   ...
   <businessObject name="PurchaseOrderBO" default="com.erpCompany.PurchaseOrderBO.java" />
</businessObjects>
```

Where: name is the layer's object name, and default is the assigned class for the layer's object.

4.1.3 Class interfaces

The use of class interfaces permit to have a controlled freedom in the customizing. The tenant can do a customization, but should respect a certain mandatory logic of the business. This business logic is provided through class interfaces (mandatory methods and attributes that should have an object responsible for some specific business tasks). If a tenant wants to customize an object, he should create a new object, which is a copy of the object to customize, and this new object should implement the class interface of the object to customize.

4.2 ERP Front-End and ERP Back-End in Multi-Tenancy Environment

Figure 4 and 5 show our suggested front-end and Back-End application in the Multi-Tenancy environment. The ERP is composed with the same layers used in Single Tenancy Environment. However, the architecture of all layers have been changed to allow an easy and efficient customizing.

A tenant can make changes on the layer's object by:

- Extending an existing object by adding new methods or attributes or by changing the implementation of the object's current methods,
- Adding new objects if the existing objects do not meet the tenant's needs.

In both cases, the customization manager will create a new object for this tenant.

4.2.1 Maintaining the customization File

To maintain the customizing in a secure and efficient way, the customization manager will do the following:

- Check if the tenant who wants to do the customizing has already his own customization file or not. This check will be done by reading the main customization file, represented as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<LayerXCustomConfig name="BOconfig.xml"
                    defaultpath="sap/frontEnd/config">
  <TenantConfigs>
    <tenant id="Tenant1" name="BOconfigTenant1.xml"
                    path="tenant1/frontEnd/config"/>
    <tenant id="Tenant2" name="BOconfigTenant2.xml"
                    path="tenant2/frontEnd/config"/>

…
    <tenant id="TenantN" name="BOconfigTenantN.xml"
                    path="tenantN/frontEnd/config"/>

  </TenantConfigs>
</LayerXCustomConfig>
```

- If the tenant does not have it own customization file, create a copy of the provided default customization file, and put it in the tenant folder.
- If a copy of default customization file is created, maintain the main customization file with the metadata of a new customization file.

Using for each tenant a customization file will improve the security and the maintenance.

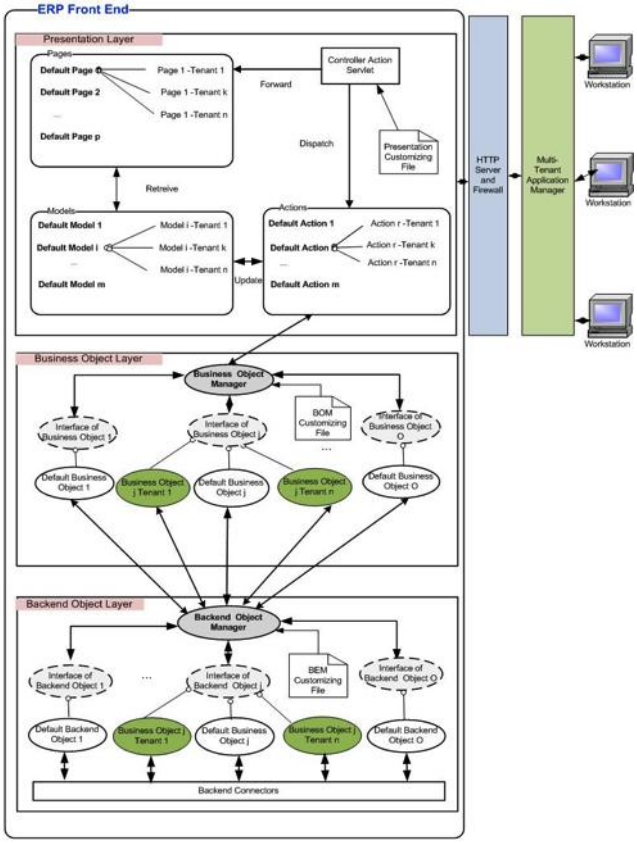

Figure 4. Proposed ERP Front-end.

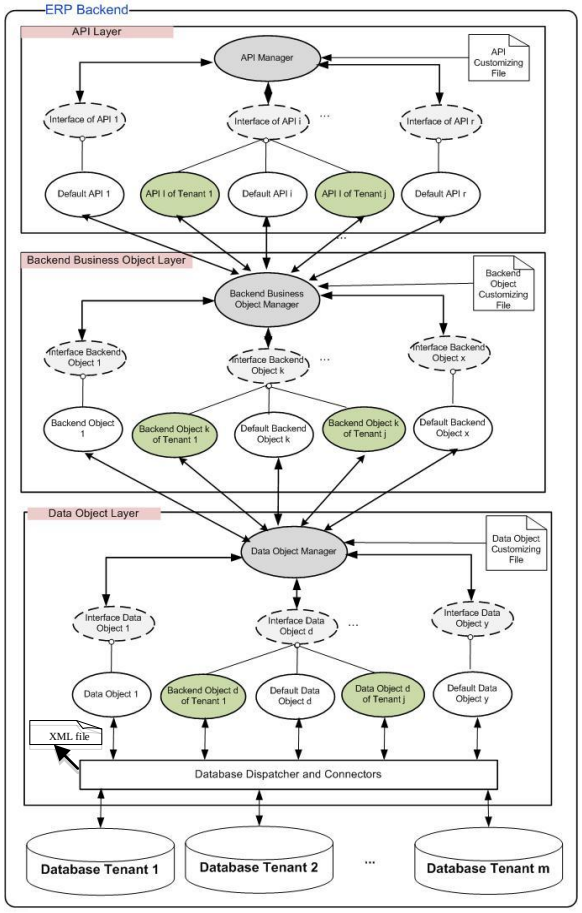

Figure 5. Proposed ERP Back-end.

4.2.2 Customizing layer objects

Customize the objects in each layer will be done as follows:

- To change an object class, customization manager makes a copy of the default provided object class and put it in the tenant folder.
- Each tenant's new object class should implement the default corresponding interface.
- If the tenant does not have its own customization file, the customization manager makes a copy of the default customization file of the corresponding layer and put it in the tenant folder
- The customization manager will update the customization file with the name of the new object class.

4.2.3 Databases

In multi tenant environment, the database can be one of the following approaches [18]:

4.2.3.1 Shared database and shared tables:

Add tenants' ID in the primary key of each database table. The tables will contain data of all tenants.
This approach has many inconvenient:

- The tables will be voluminous, since they contain the data of all tenant, this will lead to an optimization problem.
- The tenants' data are not secure; the administrators of the tenants' databases can see the data of the other tenants.

- The maintenance of the database is very complex.
- It does not support data customization, since single tables' definitions exist for all tenants.

4.2.3.2 Shared database and separated tables:

Each tenant has it own tables but tenants share the same database. This approach support data customization since each tenant has it own tables' definitions. However, this approach has the following inconveniences:

- The database will be huge, since it contains the data of all tenants, also the backend class.
- The tenant data are not enough secure, else to define a restricted access role to the administrators of the tenants' databases (only tables, views and sql program of the tenant).
- It will generate redundant data. To avoid this problem, some tables should be shared by all tenants.
- The backend class methods should be modified in order to do a dynamic sql operation, since each tenant has its own tables, the name of tables in the backend class methods will be build automatically on runtime using tenant ID.

4.2.3.3 Separated databases:

Each tenant has a dedicated Database. This approach support data customization since each tenant has its own tables' definition. Also this approach is the most secure and the easiest solution in maintenance.

In order to allow data customization in the multi tenant environment, we can adopt the second or third approach. In our framework we suggest using the third approach (Separated databases) to improve data security and providing each tenant with a dedicated database. The database dispatcher and connectors maintain a XML file that contains database names and their corresponding authentication parameters.

Note that the system uses a set of rules to restrict data customization in order to maintain database consistence and clean, such as:

- The tenant can't delete any object from the database (table, attribute, view, sql program, integrity constraint, trigger…).
- The tenant can't change the database's attribute type.
- The tenant can't change the primary key of a table but can add a secondary key.
- A consistency and dependency check is performed for any change in an object. For instance, if a tenant added some attributes in a table, a customizing wizard will invite him to update all program using insert and update on the customized table.

5 CONCLUSION AND FUTURE WORKS

Customizing ERP system in the multi tenant cloud environment considered a main issue when the companies intend to adopt the cloud ERP solution. Providing a flexible approach for customizing cloud ERP can support the companies to meet their own requirements and increase the competitive advantages between them.

In this paper we proposed a new framework that provides a managed ERP customizing approach by which the tenants will be able to tailor the multi tenant cloud ERP system with their unique requirements without affecting each other. Our proposed framework consists of front-end ERP and back-end ERP, the front-end ERP is composed from tree layers each of which interacts with the other layers to support the communication between the tenants and the back-end ERP. Furthermore, the back-end ERP is composed from tree layers that perform the business logic.

As a future work we suggest designing a user-friendly tool that supports the tenants in making changes in the system's objects, by giving recommendations and illustrating the relations among the system's objects, and showing the possible errors. Moreover, evaluating the framework is a significant work in order to study the factors that may be affected by the customizing process such as cost, security, maintenance…etc.

6 REFERENCE

[1] Engebrethson, R. (2012). Comparative Analysis of ERP Emerging Technologies.

[2] Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., & Ghalsasi, A. (2011). Cloud computing—The business perspective. Decision Support Systems, 51(1), 176-189.

[3] smbsuite.com (2012). Subscription Cloud ERP and Total Cost of Ownership (TCO). Retrieved from: http://www.smbsuite.com/SMB/media/Resources/White Papers/SMBSuite-Cloud-ERP-TCO.pdf

[4] Mattison, J. and Raj, S. (2012). Key Questions Every IT and Business Executive Should Ask About Cloud Computing and ERP. Retrieved from: http://www.accenture.com/SiteCollectionDocuments/PDF/Accenture-Key-Questions-Executive-Ask-About-Cloud-Computing-ERP.pdf [Accessed: 26 Sep 2013].

[5] Duan, J., Faker, P., Fesak, A., & Stuart, T. BENEFITS AND DRAWBACKS OF CLOUD-BASED VERSUS TRADITIONAL ERP SYSTEMS.

[6] Müller, J., Krüger, J., Enderlein, S., Helmich, M., & Zeier, A. (2009). Customizing enterprise software as a service applications: Back-end extension in a multi-tenancy environment. In Enterprise Information Systems (pp. 66-77). Springer Berlin Heidelberg.

[7] Sun, W., Zhang, X., Guo, C. J., Sun, P., & Su, H. (2008, September). Software as a service: Configuration and customization perspectives. In Congress on Services Part II, 2008. SERVICES-2. IEEE (pp. 18-25). IEEE.

[8] Jacobs, D., & Aulbach, S. (2007, March). Ruminations on Multi-Tenant Databases. In BTW (Vol. 103, pp. 514-521).

[9] Chen, W., Shen, B. and Qi, Z. (2013). Template-based business logic customization for SaaS applications.In Progress in Informatics and Computing (PIC), 2010 IEEE International Conference on (Vol.1, pp. 584 - 588).IEEE

[10] Li, Q., Liu, S., & Pan, Y. (2012, May). A cooperative construction approach for SaaS applications. In Computer Supported Cooperative Work in Design (CSCWD), 2012 IEEE 16th International Conference on (pp. 398-403). IEEE.

[11] Li, H., Shi, Y., & Li, Q. (2009, November). A Multi-granularity Customization Relationship Model for SaaS. In Web Information Systems and Mining, 2009. WISM 2009. International Conference on (pp. 611-615). IEEE.

[12] Mietzner, R., Metzger, A., Leymann, F., & Pohl, K. (2009, May). Variability modeling to support customization and deployment of multi-tenant-aware Software as a Service applications. In Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems (pp. 18-25). IEEE Computer Society.

[13] Tsai, Wei-Tek, and Xin Sun. "SaaS Multi-tenant Application Customization." SOSE. 2013.

[14] Zhu, X., & Wang, S. (2009, September). Software Customization Based on Model-Driven Architecture Over SaaS Platforms. In Management and Service Science, 2009. MASS'09. International Conference on (pp. 1-4). IEEE.

[15] Mietzner, R., Leymann, F.(2008). Generation of BPEL Customization Processes for SaaS Applications from Variability Descriptors. Services Computing (SCC), pp. 359–366

[16] Kale, N. Readings on Enterprise Resource Planning (1st ed., pp. 103-119). California: University of Southern California.

[17] Rouse, M. (2011). What is model-view-controller (mvc)?. [online] Retrieved from: http://whatis.techtarget.com/definition/model-view-controller-MVC.

[18] Sankar, R. (2013). Architecting a Multi-tenant Application. http://www.javacodegeeks.com. Retrieved from http://www.javacodegeeks.com/2013/11/architecting-a-multi-tenant-application.html

View publication stats