Improvement of an Integrated Management System for Smart Libraries Based on SaaS

Byung-Won Min

Department of Information Communication Engineering, Mokwon University, Doandong 800, Seo-gu, Daejon, 302-729, Korea
minfam@mokwon.ac.kr

Abstract

Most library systems use the client/server and ASP services for their software. However, such systems are difficult to manage and incur high operating costs because of the problems related to high HW and SW purchasing costs, installation and distribution, customization, upgrade, fault and problem management, and costly license royalties.

To solve such problems, this study developed the key elements that can be deployed in the SaaS-based digital library system supporting the multi-tenant environment (SaaS maturity level 3 or higher).

The system was deployed with the SaaS-based software on-demand type service model, which requires little initial investment, is simple and easy to use, and delivers the IT service at a low cost.

The system proposed in this paper features multi-tenant-oriented application resource sharing and optimization, multi-tenant-oriented data space sharing and isolation, multi-tenant-oriented back-end data management, and isolated multi-tenant-oriented hosting in order to solve many problems of the existing systems.

Keywords: Smart Library System, Integrated Management System, Multi-Tenant, SaaS

1. Introduction

The latest trend of the software market includes cloud computing, SaaS (Software as a Service), utility computing, SOA, Web 2.0, and RIA. What these technologies share in common is that they are all services which together have become the key growth engine of the market [1]. As such, the current focus on SaaS technology, which is transforming the software into a service, is natural.

SaaS technology is related to virtualization and cloud computing, which were ranked number one and two, respectively, in the top ten strategic technologies in 2009 by Gartner. SaaS is an application virtualization technology and one of the key elements of cloud computing [2]. While SOA, Web 2.0 and RIA represent the change in the way of developing, operating and using the software, SaaS represents the change in the way of distributing the software [3]. SaaS can be defined as an up-to-date software distribution model that allows users to use software as an online service over the Internet [4, 5]. Moreover, it can be also defined as a technology for distributing application software to multiple users via the online service. It frees users from the burden of complex software and hardware management, as they can use the software over the Internet and only pay for what they use.

The SaaS business model requires very little initial investment from the customer and no management of the system. Customers can pay a fixed fee for a certain service period or pay in proportion to their usage. SaaS can be divided into hosted application management, which
supplies packaged applications, and software on-demand, which provides the software and supports multiple users over the Internet [6].

Currently, most library systems use the client/server and ASP services for the software. However, such systems are difficult to manage and incur high operating costs because of the problems related to high HW and SW purchasing costs, installation and distribution, customization, upgrade, fault and problem management, and costly license royalties.

The SaaS-based integrated digital library system proposed in this paper is different in that it customizes using the metadata and supports the tenants, representing the users and user groups, with a software instance. It also solves the weaknesses of ASP, which is not only costly to customize but also cannot take advantage of economy of scale because it loads each instance individually.

Furthermore, there is very little initial investment since the library functions are standardized and modularized. They are designed and deployed as the cloud-based software on-demand type service model so as to enable easy, simple and low cost IT services.

Following the introduction, Chapter 2 discusses the design and architecture of the system; Chapter 3 presents the procedure for developing the system; Chapter 4 presents the results of the performance analysis of the proposed system; and, lastly, Chapter 5 presents the conclusion of this study and a possible direction for future studies.

2. Integrated Smart Library System Design

2.1. Related Studies

2.1.1 SaaS: The Compared to the conventional ASP, SaaS is similar in that users use the software over the Internet, but differs in that users can customize the software, whereas with ASP the software provider performs that function. From the software supplier’s position, it is different in that it uses metadata for customizing and supports the tenants, representing the users and the user groups, with a software instance. It solves the weaknesses of ASP, which is costly to customize and cannot realize the economy of scale because it runs the instances individually. In summary, SaaS software has following characteristics [7].

- Commercial software is provided via a network-based approach and management
- System is managed at a center rather than a customer site, and users access the software through the Web
- Application delivery is closer to the 1:n model than 1:1. It includes the architecture, price, partnering and management attributes.
- Centralized function update is provided to eliminate the needed for patch or upgrade download.

The maturity of the SaaS architecture can be categorized as follows according to its ease of configurability, multitenant efficiency, and scalability [8-12].

- Level 1: Ad Hoc/Custom
  It has a similar form to that of the conventional ASP service. As it provides a customized application instance to each customer, it is difficult to manage and costly.
- Level 2: Configurable
  An application instance is provided for each customer. Although customization is possible by configuring the service, it does not support multi-tenancy. The supplier must provide sufficient hardware and storage to support the large volume of application instances.
- Level 3: Configurable, Multi-Tenant-Efficient
It supports all customers with one instance and uses the metadata for customizing. It provides multi-tenancy. The supplier can manage the resources more efficiently compared to level 2.

- Level 4: Scalable, Configurable, Multi-Tenant-Efficient
  This is the most mature model, supporting all customers from a load-balanced farm. It balances the customer data for management and supports customizing with reconfigurable metadata.

Multi-tenancy in SaaS is a software architecture principle of which a single instance of the software is run in the SaaS provider's server to support multiple tenants [13-16]. Sharing of the lower-level computing resources by the tenants is an essential element of cloud computing. Such sharing of computer resources by logically separated tenants is also called multi-tenancy [17, 18].

We pictorially present overview of SaaS as shown in Figure 1.

![Figure 1. Overview of SaaS](image)

### 2.2. Integrated Smart Library System Architecture

The integrated smart library system developed in this paper to support a SaaS-based multi-tenant environment is composed of a multi-tenant application service layer, a SaaS common service layer, and a SaaS application generation environment layer, as shown in Figure 2.

The multi-tenant application service layer performs the multi-tenant metadata management function, legacy service interface function, multi-tenant metadata security function, and multi-tenant application execution environment management function; and consists of metadata interface single-tenant application execution technology, metadata interface multi-tenant application execution technology, and multi-tenant load balancing technology.
The common service layer is an open API, and performs the customer management function, billing management function, access management function, service level agreement (SLA) management function, metering function, and logging function. It applies SaaS platform customer management and billing technology, SaaS service access control and SLA monitoring technology, SaaS service metering technology, and SaaS logging technology.

The application generation environment layer performs the data schema management function, business logic management function, SaaS application technology language and development environment function for the user interface, metadata model management function, and code generator. The applied technologies include the SaaS application description language definition, SaaS metadata model definition, SaaS code generator, SaaS application development environment, and SaaS platform interface testing environment technology.

2.3. Conceptual Diagram of Service

The service components of the integrated digital library system developed in this study consist of the service library component, academic research data integrated repository component, management library component, legacy system interface component, data center interface component, and user and outside agency component.
The service library component performs the service pooling functions to deploy the services of integrated search, browsing, academic journal and article search, meta search, and My Library.

The in-house academic research data integrated repository component performs report management, check-in management, article index management, binding management, original document management, resource management, integrated index management, and server DB management using the SOAP/WSDL, XML/MARC, and AJAX technologies.

The management library component function performs document security, PDF conversion, natural language index, book receiving, catalog, inspection, periodicals, article index, and SDI management. The legacy system interface component functions include SSO (Single Sign On) with the portal and interface with the groupware, knowledge management, HR management and CD_NET systems. The data center interface component performs the meta search engine function and the NOS (NDSL on Site) function to interface with Web DB, Internet bookstores, digital journals worldwide, and academic portals worldwide[20]-[22].

2.3. Development Environment

The SaaS platform-based integrated digital library consists of three layers as shown in Figure 4. The first layer provides the Nereus API in the SaaS platform; the second consists of
Spring Framework, AOP, ORM, DAO, MVC, Web, Context, etc.; and the third layer, which is designed for application development, consists of Web (JSP, JSTL, HTML, CSS, JavaScript, and XML), Business Logic (book receiving, catalog, search, article index, continuous periodicals, etc.), iBatis, and Oracle.

Spring Framework is an open source framework created to support the development of complex enterprise applications. One of the strengths of Spring Framework is that it is a layered architecture. Its advantages include the fact that a consistent framework is provided for J2EE application development and that users can selectively use only the desired components simultaneously.

iBatis enables object orientation by separating the business logic and database access. It ensures flexibility of data processing, as the SQL statement and code can be independently generated. It also provides the transaction and cache functions.

![Figure 4. Development Environment Schematic Diagram](image)

### 2.4. Functional Class Architecture

The structure for developing the applications of the SaaS platform based integrated digital library system is largely composed of three classes as shown in Figure 5. The controller class controls the Web screen flow, calls service classes and processes views (JSP). The service class deploys the business logic, inspects data validity, handles transactions and calls the repository class. The repository class performs SQL and business logic for the database.
3. Integrated Smart Library System Development

3.1. LinkSaaS Application Platform

LinkSaaS Application Platform is software infrastructure designed to tame the data deluge with cost-effective, supercomputer-like capacity to process, transform, and indeed channel it for our benefit. It does this by leveraging the spare capacity of computing resources on the “edge” of the Internet – resources which generally use only a small fraction of their capacity even when busy and are available for marginal operating cost and environmental impact – and allowing users and developers to combine and coordinate them in novel, dynamic ways. In order to assure the safety of these edge resources, LinkSaaS Application Platform builds upon Java applets, the best established security context for running untrusted, third-party code on the Internet; the use of this technology also seamlessly extends the reach of LinkSaaS Application Platform across different platforms, operating systems, and even enterprises. In principle, the LinkSaaS Application Platform network could encompass all the world’s network-attached computers, providing an aggregate capacity 100 times greater than the 500 most powerful supercomputers combined.

The purpose of the LinkSaaS architecture is twofold: first, LinkSaaS is designed to increase the productivity of the installed base of computing devices. As noted above, most of the computers in the world use very little of their computing capacity even when “busy”, e.g., editing a letter or spreadsheet, or browsing the Web. LinkSaaS aims to harness this spare capacity for general computing. However, there are significant technical obstacles to realizing this purpose. For this reason, LinkSaaS is designed to be simple to join, leave, and administer, with the resource owners remaining firmly in control of their computers; moreover, it must satisfy reasonable security expectations for running untrusted, third-party code.

Second, LinkSaaS is intended to increase the productivity of developers creating new applications for this large network. Distributed programming historically has been problematic and fraught with tedious details – in short, not usually worth the effort except in extreme cases. The easiest situation is where the developer only has a single computer cluster under his influence, but this cluster may not be large enough. Once administrative boundaries are crossed, even within the same organization, the developer must now keep track of a variety of configurations, accounts, and security mechanisms. In addition, the code usually has to be built and installed on platforms which all differ, even if only slightly (hence the
tedious details). The code then has to be activated, and the network configuration itself, such as the presence of firewalls, must be considered.

LinkSaaS addresses the above problem by specifying a streamlined environment for programming and deploying code across the network based on Java applets. This environment will be described in greater detail below. With LinkSaaS, developing, deploying, and provisioning a large-scale network application becomes realistic and practical.

3.1.1. Description: In this section, the LinkSaaS architecture is described. Its components are described first, followed by descriptions of how developers, users, and administrators see the resulting system. Every attempt has been made to keep LinkSaaS simple and straightforward to join, leave, program, and administer.

The LinkSaaS network itself provides a basic infrastructure which makes the network programmable much like a peer-to-peer network while still respecting administrative constraints. It is not intended to make a distributed system look like a single computer; the authors suggest that this level of abstraction is best approached through software built on the LinkSaaS infrastructure. Moreover, the single computer abstraction is neither necessary nor appropriate in many circumstances; instead, programmers should be free to use the network creatively, and to construct network configurations appropriate to the nature and scale of the problems they are tackling.

To the extent that the LinkSaaS architecture is based on services which can be combined to form other services, it will be seen to bear some relation to a service-oriented architecture (SOA). However, LinkSaaS services do not have to reside on big, expensive servers; instead, they are provisioned by resources on the edge of the network. Also, unlike a typical SOA, LinkSaaS specifies a streamlined remote service deployment model, facilitating dynamic, highly distributed service provision; it thus relieves the programmer of some of the most tedious aspects of practical distributed programming.

Furthermore, LinkSaaS relies on the Java Virtual Machine (JVM), with its built-in security mechanisms, to accomplish remote deployments safely; it should be noted that Java applets have been running untrusted, third-party code in web browsers for over ten years. The JVM allows compiled code to run under Windows, Linux, Mac OS, and a number of other, less familiar operating systems, as well as on hardware ranging from the typical x86-based PC to an ARM-based mobile phone. The speed penalty for running code in the JVM has also largely vanished due to HotSpot runtime compilation into native machine code. By building on Java technology, LinkSaaS can offer secure, high-performance, cross-platform computing.

3.1.2. Components: Figure 6 shows a block diagram of a LinkSaaS network. The LinkSaaS network is divided into administrative domains, each of which can have one or more LinkSaaS servers. Each server is in turn connected to the Internet, through which it communicates with other servers as well as generic web servers. LinkSaaS clients then connect to one or more LinkSaaS servers; in principle, they could connect to servers in different domains, if their network access allows it.

It can be seen that the LinkSaaS network is an overlay network built, firstly, on the Internet itself. LinkSaaS servers are identified publicly by their Internet addresses, and since LinkSaaS servers and domains are fundamentally independent of one another, the LinkSaaS network is scalable in the same sense that the Internet itself is scalable. The network within a LinkSaaS domain may be different, however, and the clients may or may not have direct access to the Internet – for instance, because of a firewall, or its implementation on a proprietary network. Even so, the LinkSaaS clients will remain accessible to LinkSaaS network operations.
3.1.3. **LinkSaaS Server**: The LinkSaaS server is the lynchpin of the LinkSaaS network: clients join the LinkSaaS network by connecting to a server, and the server acts as a proxy for the clients in communication with the wider network as well as with other clients (Figure 7).

In this way, the clients can communicate with the wider network even if they are protected behind a firewall – only the server need be exposed to the wider world. Within an administrative domain, a LinkSaaS server should be maintained and monitored as any central service, such as an e-mail service or an institutional web server. In turn, because of its central role in the LinkSaaS network, the server can act as an administrative control point for its clients, monitoring LinkSaaS bandwidth usage and limiting it if necessary.

Thus, even though clients can communicate with one another as if they were participating in a peer-to-peer network, the administrator can maintain a useful degree of control over their resource usage. The LinkSaaS architecture therefore addresses the common shortcomings in peer-to-peer network operation.

The LinkSaaS server manages the namespace of its domain, or, in other words, it assigns names to the clients connected to it. These names are globally unique by construction as long as LinkSaaS servers maintain unique IP addresses – which is an assumption of the Internet itself.
3.1.4. **LinkSaaS Client**: LinkSaaS clients are the workhorses of the LinkSaaS network: they are the programmable elements out of which developers create applications. The client itself is a program which runs on a host computer: it can be started from a web browser (applet client), run as a user program (stand-alone client), or installed as a persistent service (enterprise client). It is possible to have several clients running on a single computer; each client will have a unique identifier with which it identifies itself to the LinkSaaS server.

As noted above, a LinkSaaS client downloads its first service descriptor from a LinkSaaS server. This service descriptor tells the LinkSaaS client what to install in its root container. In the case of a stand-alone or enterprise client, the client can connect to more than one server simultaneously, initializing a root container with a different service descriptor for each – since each LinkSaaS server, potentially in different administrative domains, may implement different domain policies. It is the responsibility of the LinkSaaS server to decide whether or not to accept a connection from a particular LinkSaaS client.

The security context of an applet client is shown in Figure 8. In this case, the entire LinkSaaS client is encapsulated within a Java applet and its Security Manager.
3.1.5. **LinkSaaS Application Server:** LinkSaaS clients are able to access normal web resources via the LinkSaaS server. This feature allows them to access not only the whole World Wide Web for information or other functionality, but it also allows the developer a convenient method to control his application.

As noted above, LinkSaaS does not specify a particular application model, but a particularly simple model, used already for several applications, consists of a web server (the application server) which provides a work queue and a user interface through which a user submits to it. Clients are set up to poll the application server with simple HTTP GET requests for work to do. Results can be sent back via HTTP POST operations or stored until fetched and cleared by the application server.

The application server is not a part of the LinkSaaS network per se, and indeed can be written without any reference to LinkSaaS code; its interaction with LinkSaaS clients can be managed entirely through normal web and socket connections made through the LinkSaaS server. The server therefore could be implemented using simple CGI scripts on an existing web server, though more flexibility may be obtained by using streamlined web server infrastructure software such as Grizzly or LinkSaaS’s own NIONetworkHandler class. It should also be noted that a LinkSaaS service could also function as the application server.

3.1.6. **LinkSaaS Service:** A LinkSaaS service is a program that can be installed and run on a LinkSaaS client. Services can perform data processing, serve web content, and communicate with other services or external web sites. In addition, a LinkSaaS service can create child containers into which other LinkSaaS services can be installed.

It is worth noting the terminology: a LinkSaaS service runs on a LinkSaaS client, not a server. However, it can be said that a LinkSaaS server (which after all provides the root of the

![Figure 8. LinkSaaS Applet Client Security Context. The Service is Prevented from Reading Local Files or Making Arbitrary Internet Connections](image-url)
service’s public URL) is what offers the service – even though the actual execution of the service is on another computer.

A LinkSaaS service is initialized by pointing the client to a web-accessible service descriptor in the form of an NML document. This descriptor can be located on any web server, not necessarily the application server or any LinkSaaS server (though the NML document describing the root container resides on the LinkSaaS server by design).

The LinkSaaS Markup Language (NML) has an XML-like syntax, with three tags defined:

- `<LinkSaaS-app>` defines the application’s name and encompasses all the classloaders and services it uses.
- `<classloader>` defines the location of class hierarchies and JAR files for services defined within its scope.
- `<service>` defines an actual service with a name and a main class. Data between the opening and closing tags are passed to the service’s initialization method as a character sequence (CharSequence).

The `<classloader>` tag is optional; any services defined outside a `<classloader>` scope is assumed to find classes starting from the service descriptor’s URL directory. It is often convenient for any necessary class or JAR files to reside alongside the service descriptor, but the `<classloader>` tag allows applications to draw upon classes from a variety of sources.

An example of an NML document describing a CommandService, an often-used root container, is shown below.

```xml
<--! Last touched 18-03-08 -->
<nerus-app default="command">
  <service id="command" code="org.nerus.commandservice.CommandService">
    a7a3b5bc14a48f6a6e4af57ba0f0c7
  </service>
</nerus-app>
```

**Figure 9. NML Document for CommandService Descriptor**

In this example, the application is called “command”, and its Java classfile is found relative to the server’s default web directory. The data between the `<service>` tags (in this case a primitive authorization string) is passed to the service on initialization.

An NML document can describe multiple services with different code sources. Services may be combined into larger services by including references in the initialization data, whether to other service names within the same NML document, or using URL’s of external services. An “auto-configuration” LinkSaaS service may be employed to generate further service descriptions as needed.

**3.1.7. LinkSaaS Api:** The API of a LinkSaaS service is much like that of a Java applet: if it is deployed in a standard JRE, then it can use any of that JRE’s libraries. However, the usual applet security restrictions apply: it cannot modify system properties, cannot access the local filesystem directly, and cannot open network connections except to the server from which the applet was originally downloaded. Actions beyond these must be authorized by the client (and by implication, the user) through Java’s extension mechanism.
The most basic LinkSaaS service interface (LinkSaaSService) specifies only an init() entrypoint, which is called when the service is created. The name of the service is determined by the service descriptor; the client delegates URL’s with this path element to the service.

An AbstractService class has been provided to make it easier to write new LinkSaaS services. AbstractService provides two abstract entrypoints, prepareToListen() and handleConnection(). prepareToListen() is invoked when the service is created (it is invoked by the AbstractService.init() method). handleConnection(), on the other hand, is called whenever a connection is made to the service from the outside, including the times when a user clicks into the service through the LinkSaaS server.

Because of the LinkSaaS client’s applet-like constraints, several methods have been provided in AbstractService to facilitate communication with the outside world. First, println() and printErr() methods have been provided to print messages to the output and error panels in the LinkSaaS client’s user interface. Web access must be redirected through the LinkSaaS server, which serves as a web proxy for its clients: a createRedirectedURI() method is provided to create the proxy URL from an original URL. A socket proxy is also provided.

The common application pattern described above is implemented on the service side by using prepareToListen() to create a new thread which periodically polls the application server (whose web address has been transformed by createRedirectedURI()) for work to do. In this case, handleConnection() can be used by the application server to fetch stored results, or by other LinkSaaS services to fetch intermediate results. It can also respond to requests for status information, for instance by users clicking through the LinkSaaS server web page.

3.1.8. LinkSaaS network and namespace: One of the key benefits of the LinkSaaS network to network programming is to define a straightforward namespace by which elements of the network can be addressed. The LinkSaaS namespace consists of URL’s, all of which are valid for public web access.

Each LinkSaaS server has a public URL such as http://domain1/, where “domain1” refers to the LinkSaaS server host. Since a LinkSaaS domain will typically have one or a few LinkSaaS servers, and policies are determined on a domain level and implemented by the servers, it is appropriate (though not necessary) to designate LinkSaaS servers with domain names. Since the LinkSaaS servers are publicly accessible Internet nodes, their names must be registered in the normal way.

The LinkSaaS server provides several web pages, such as http://domain1/clients, which lists the clients connected to it.

When clients connect to a LinkSaaS server, it creates a root container into which the default service, described by DomainRoot.nml, is installed. This service is addressed by appending the client alias to the clients subdirectory of the server URL, e.g., http://domain1/clients/client5. The client then delegates the interpretation of any further path elements to the service. Thus, if the service has created child containers in which it has installed services, those services are referred to in further path elements.

In principle, the entire LinkSaaS namespace is accessible via the Web. If a service needs to address another service, it creates a redirected URL out of the other service’s public URL, just as if it was a normal web resource.
3.1.9. LinkSaaS Application: Software delivered as a service offers distinct advantages over software delivered by more traditional means: it is frequently mobile, web-based, centrally managed, and nearly free of troublesome installations and patches. In its simplest form, the architecture implied by these features consists of many clients attached to a large server infrastructure. Increasing numbers of users as well as improved functionality generally impact the server load and therefore drive server requirements, with increasingly expensive outcomes. Reducing server load therefore decreases the overall cost of the system, but risks substantially increasing operational costs.

![Figure 10. Interactions within a LinkSaaS Application](image)

3.2. Development Scope

The system proposed in this paper consists of a total of twenty-two modules including twelve for the Web-based library automation system, four for electronic source management, three for search engine, and three for language resources and others.

The Web-based library automation system is the module that performs the main functions of the integrated electronic library system, which include received book management, catalog management, check-out/return management, periodicals system, the article index system, the integrated search system, the original document copying service system, a personalized information service, the research report management system, and the contents management system.

Electronic source management includes NOS (NDSL on Site), BOS (Books on Site), ERM (E-Resource Management), and IRS (Institutional Repository solution for dSpace), etc.
3.3. Service Scenario

The SaaS platform-based integrated library system is divided into the service for the administrators and the service for the users (publishers, contents providers, cross reference, third-party users, etc.) as shown in Figure 12.

The services for the administrators include the registration, modification and removal of received books and check-out information in the database using the administrator module, such as the received book system, catalogue system, inspection system, article index system, periodical system, VOD system, and original document management system.

The services for users include the search service, inspection service, original document copying service, updated information delivery service, and searched data management service.
3.4. System Development Environment

The development environment for a smart library information service system that supports the multi-tenant environment is shown in Table 1. It uses Linux5.0 as the operating system, Intel Xeon Dual Core Processor 2.0GHz CPU, 2GB ECC DDR SDRAM memory, and a 146GB/10k rpm SAS Disk * 4 internal disk.

Table 1. Development Environment

<table>
<thead>
<tr>
<th>Item</th>
<th>Detailed Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
<td>• OS: Linux 5.0 or higher</td>
</tr>
<tr>
<td></td>
<td>• Type: Intel Xeon Dual Core Processor</td>
</tr>
<tr>
<td></td>
<td>• Clock Speed: 2.0GHz or higher</td>
</tr>
<tr>
<td></td>
<td>• Quantity: 2 or more (up to 2 expandable)</td>
</tr>
<tr>
<td>Memory</td>
<td>• 2GB ECC DDR SDRAM (8GB Memory or more) or more</td>
</tr>
<tr>
<td>Internal Disk</td>
<td>• Capacity: 146GB/10k rpm SAS Disk * 4 or higher</td>
</tr>
<tr>
<td></td>
<td>• Disk Bay: 6 or more</td>
</tr>
<tr>
<td>IO Slot</td>
<td>• 6 Hot-Swap PCI-X 64bit slots or more</td>
</tr>
<tr>
<td>LAN</td>
<td>• Dual 10/100/1000Mbps Ethernet Controller(TP RJ-45)</td>
</tr>
<tr>
<td>Graphic-Card</td>
<td>• VIDEO: 8MB SDRAM or higher</td>
</tr>
</tbody>
</table>

3.5. Multi-tenant Environment

Multi-tenant technology supports the sharing of an instance by multiple customers. In the SaaS platform, multi-tenancy is the function that allows then individual user (user organization or tenant) to configure the environment.
Figure 13 shows the multi-tenant architecture deployed in this study. The tenants were grouped in Nereus client, and a Nereus server managed the tenant environment. Therefore, the computer resources were allocated, and the multi-tenants connecting the platform and application codes at the same time were supported by a single instance.

![Multi-tenancy Architecture](image)

**Figure 13. Multi-tenancy Architecture**

### 3.6. Development Details

Centered on the system development scope and service scenario concept described above, the digital library system supporting the SaaS-based multi-tenant environment developed in this study consists of 22 sub-systems and lower level modules. The functions of these subsystems are described as follows:

1. **Administrator Application**

   The key design concept of the administrator application shown in Figure 14 is customization. The screen was designed to enable users to edit the data at the top of the screen for each category (menu). The overall color concept is green, which gives out a subtle impression and imposes a lesser burden on the eyes, thus reducing fatigue.

   Although the design element is important, the newly renovated site focused on usability and accessibility to enable users to easily find information, and on the development of a system that conforms to the Web standard so that the users can login from all computers including tablet PCs. Area ① of Fig.1 is where updates occur frequently. It displays the main company business with the image and simple description so that the homepage gives out the impression of a dynamic rather than a static site. Area ② is designed to be a blog format to show the image and brief description of new or ongoing projects so that users can see more information on the initial screen. Therefore, tablet PC users can find the information they need right in the first screen. Moreover, since SNS (Social Network Service) is an essential element of the site promotion, Twitter and Facebook icons were included in the updated data. Area ③ shows the postings in Linksoft Twitter to enable efficient interaction between the site and users.
2. User Application

The key design concept of the user application is customization. The screen was designed so that the administrators can edit the data at the top of the screen for each category (menu). In other words, the administrator (user) can edit the data shown according to the interest or business importance.

The main menu at the top of the screen was designed with icons so that the administrators can easily recognize the menu, and right/left scrolling was provided so that the menu can be flexibly added to or removed.

The table format was designed to give out a refined and advanced impression instead of the familiar conventional table design. The design concept was centered on maintaining consistency with 1Depth design and on using CSS for input forms of detail view.

4. Analysis of the Performance of the Proposed System

To measure the performance of the proposed digital library system supporting SaaS-based multi-tenant environment, a simulation environment consisting of ① Web server, ② DBMS server, ③ media server, ④ exchange server, ⑤ IPPBX server, ⑥ client 1, and ⑦ client 2 was constructed, as shown in Figure 15.

IIS 6.0 is used as the ① Web server in which the proposed server module is installed. The proposed server module and MSSQL 2005 were used for the ② DBMS server. The proposed server module and Adobe Media Server 3.1 were used for the ③ media server.
The proposed server module and Red5 Media Server 0.7.0 were installed in the exchange server. The proposed server module and Asterisk 1.4.21 PBX were installed in the IPPBX server. The proposed client module, Internet Explorer 6.0, and application programs such as MS-Office 2007, Haansoft 2007, and ViRobot Desktop v5.5 were installed in client 1, while the proposed client module and Firefox 3.0 were installed in client 2 for the client module test.

To measure the performance, TeamQuest 10.1 Manager was installed in the Web server and TeamQuest 10.1 View in client 1. The programs measured the server resource usage. LoadRunner 8.1 was installed in client 1 to test the load generation.

![Diagram of system setup](image)

**Figure 15. Analysis of the Performance of the Proposed System**

### 4.1. Efficiency Analysis

The server resource utilization rate, server memory usage, and server response time were measured when 100 concurrent users logged in to the proposed digital library system to perform the registration, modification, removal, and search of data. The resource utilization rate, represented in %-processor time, indicates the percentage of time spent on running non-idle thread. Memory usage, represented in ‘private Mbytes’, is the amount of memory allocated to the process run by the computer. Server response time is measured in terms of the number of seconds it took to complete the command from the time a command such as a system search or order command was input.

Figure 16 shows the results obtained for the server resource utilization rate (%) and average memory usage of the server under the given analysis scenario. Although the maximum resource utilization rate of the server could temporarily increase to up to 7.81% under the conditions of scenario 1, it returned to the stable condition after the function was executed. The average server memory usage was measured to be around 803MB. The CPU and memory resources of the proposed system showed very stable utilizations to run the analysis scenario, and the data management was also very stable under the set condition.
Figure 16. Result of Efficiency Analysis

Figure 17 shows the system response time under analysis scenario 1. The average response time obtained for the four measurements of registration, modification, removal, and search of the different data was 62.9 seconds, indicating an average response time of 0.629 seconds per client. Such a response time is the level required to assure real-time play of the contents and may be considered very satisfactory.

Figure 17. Time Efficiency according to Data Change

To analyze the system’s efficiency, a scenario consisting of ‘measuring the server resource utilization rate, server memory usage and server response time after logging in an digital library system designed for 100 concurrent users and performing the registration, modification, removal, and search of the data’ was devised. The resource utilization rate, represented in %-processor time, indicates the percentage of time taken to run a non-idle thread. Memory usage, represented in ‘private Mbytes’, is the amount of the memory allocated to the process run by the computer. The server response time is
measured in terms of the number of seconds it took to complete a command from the time a command such as system search or order command was input.

The figure below shows the result obtained for the server resource utilization rate (%) and the average memory usage of the server under the analysis scenario. Although the maximum resource utilization rate of the server could temporarily increase to up to 7.81% under the scenario-1 condition, it returned to stable conditions after the function was executed. The average server memory usage was measured to be around 803MB. The CPU and memory resources of the proposed system showed very stable utilizations when running the analysis scenario, and data management was also very stable under the conditions.

The results of performance tests, from searching text without the benefit of an index, are shown in Figure 18. It is encouraging that, at least up to 100MB, the job time is approximately linear and with only a moderate overhead of less than one second.

![Figure 18. Job performance of Smart Library application](image)

4.2. Analysis of Performance Improvement of the Proposed System

The proposed digital library system supporting the SaaS-based multi-tenant environment improved the four functions described below.

- Multi-tenant-oriented Application Resource Sharing and Optimization

Although the processes must be run in the shared memory and the process space for the benefit of multiple tenants, the visible memory, process status, metadata configuration, performance change, and handling of incorrect errors among the tenants must be isolated for protection.
• Multi-tenant-oriented Data Space Sharing and Isolation

In the application platform the technology layer is separated by the data repository. The data layer is deployed to assure multi-tenancy. The multi-tenancy in the data level is automatically deployed by the data repository or designed and separated by the application.

• Multi-tenant-oriented Back-end Data Management

As the database for just one user no longer exists, the point of data recovery cannot be determined. Therefore, the system proposed in this paper strictly applies multi-tenant orientation for data backup and recovery, problem, rollback, roll-forward, diagnosis, import/export, and other DBA-oriented back-end database processes.

• Isolated Multi-tenant-oriented Hosting

The application data and business logic of a tenant must be protected from other tenants. Therefore, the host provider must support all the operations of the application, and the tenant is assigned a function for accessing the business data of the host DBA of other host tenants.

• Multi-tenant-oriented Security, Monitoring, and Reporting Management

The users affiliated with a tenant should not have access to the resources outside of the tenant and should be provided with isolated monitoring and independent management elements and a policy specific to the tenant.

5. Concluding Remarks

Most library systems use the client/server and ASP services for their software. However, such systems are difficult to manage and incur high operating costs because of the problems related to high HW and SW purchasing costs, installation and distribution, customization, upgrade, fault and problem management, and costly license royalties.

To solve such problems, this study developed the key elements that can be deployed in the SaaS-based digital library system supporting the multi-tenant environment (SaaS maturity level 3 or higher).

The system was deployed with the SaaS-based software on-demand type service model, which requires little initial investment, is simple and easy to use, and delivers the IT service at a low cost.

The system proposed in this paper features multi-tenant-oriented application resource sharing and optimization, multi-tenant-oriented data space sharing and isolation, multi-tenant-oriented back-end data management, and isolated multi-tenant-oriented hosting in order to solve many problems of the existing problems.

However, the digital library system applications developed with JavaEE or the .net application platform use simple or isolated tenancy model distribution and could not satisfy the requirements of the SaaS scenario. As such, further studies will be required in that area in the future.

References


Author

Byung-Won Min

He received M.S degree in computer software from Chungang University, Seoul, Korea in 2005. He received Ph.D. degree on Department of Information Communication Engineering from Mokwon University, Daejeon, Korea. He worked as a professor in the department of computer engineering, Youngdong University, Youngdong, Chungbuk, Korea, from 2005 to 2008. He joined the faculty of the Dept. of Information Communication Engineering, Mokwon University, Daejeon, Korea, in 2008. His main research interests include Ubiquitous Sensor Network and embedded system. Cloud Computing. Smart Library, SaaS