Configurability in SaaS (Software as a Service) Applications

Nitu
MCSI, MIETE
Scientist ‘D’
ISSA/DRDO, Metcalfe House, Delhi,
India.
Telephone No.+91-11-23882158.
nitu2805@gmail.com

ABSTRACT

In the last few years Software as a Service (SaaS) has changed from curiosity generating concept to an accepted mainstream concept. SaaS has transformed the way software is being delivered to the customer. This transformation has been possible with maturing technologies and the web becoming the primary medium of communication and collaboration. SaaS is a software delivery paradigm where the software is hosted off-premise and delivered via web to a large number of tenants and the mode of payment follows a subscription model.

Configurability is one of the keystones to the success of any SaaS software**.** Configurability allows the single instance multiple tenant model which leads to many benefits both for the customers and the vendors which in turn has led to the acceptance and popularity of SaaS. Configurability in SaaS software aims to provide tenants/customers with a multitude of options and variations using a single code base, such that it is possible for each tenant to have a unique software configuration. How to capture and represent configuration data is an important consideration when designing for configurability in SaaS software.

This paper addresses the issue of how to effectively and efficiently support configurability in SaaS software and proposes SaaS architecture to support configurability. A proof of concept implementation is done to support configurability with the configuration data in xml format. The paper aims to provide information on the nature of configurability in SaaS software, how it can be provided and the technologies needed to support it.

Categories and Subject Descriptors

D.2.11 [**SOFTWARE ARCHITECTURES**]: Software as a service

General Terms

Design, experimentation.

Keywords

Software as a Service, SaaS, Configurability, .Net

1. INTRODUCTION

Since early 90’s there has been a rapid advance in the internet in terms of speed, connectivity and reliability. With the internet becoming ubiquitous, coupled with the vendor’s interest in capturing the market of small customers who could not afford the expensive enterprise software, either on-premise or through ASP, fuelled the advent of Software as a service (SaaS). Software as a Service (SaaS) is a software delivery paradigm where the software is hosted off-premise and delivered via web and the mode of payment follows a subscription model.

The term Software as a Service as camel back acronym SaaS can be traced back to a white paper called " Software as a Service: Strategic Backgrounder" by the software & Information Industry Association's eBusiness Division published in Feb. 2001[14] though WebEx meeting hosting was launched as an on-demand service in 1999 and Salesforce.com was launched in February 2000. The earlier SaaS adopters were stand alone, multi-tenant applications having limited configurability with a focus on rapid deployment and lower total cost of ownership for the customer.

At this time SaaS model seemed very similar to Application Service Provider (ASP) model, but there were many subtle differences between the two. ASPs, which flourished in 1980s and 1990s, hosted third party client server applications in a data centre kind of environment. Since each ASP had many customer-specific applications running, it couldn't provide much expertise in each application. Customers still had to have in-house expertise to make sure the applications were behaving correctly. Also building and maintaining datacenters was a costly affair for the ASPs and the high cost filtered to the customers.

A typical SaaS provider offered applications specifically designed to be hosted and delivered over the Internet to many customers. With SaaS, the customers' IT people had access to a number of different features or capabilities within the software set and could fine-tune them via a Web interface to fit the needs of their company. A true SaaS model adhered to the single instance multiple tenants model of application delivery. The fine tuning which is know as configuration allowed this single instance multiple instance model. Thus, even though there was single instance of the software running, the customers got a unique application depending on the fine-tuning/configuration done by them. Since SaaS applications were centrally managed/ administered by the vendor, there was no need for in-house application developers or experts at customers end. Maintaining a single code-base was simpler and cheaper for the SaaS vendor which in turn led to faster upgrades and cost savings at the customer end.

It was only in 2005-2006 that the SaaS wave gained momentum. The reason for this was that not only the internet had become high-speed and affordable, but also the customers had started becoming comfortable with doing business on the web. It was realized that good backup and fault tolerant practices at a good SaaS vendor may make data more secure and reliable with the SaaS vendor rather than with own enterprise. Since the number of SaaS vendors had increased, the customers had the advantage of flexibility in making their IT decisions as they had not invested in the infrastructure and could always shift to another vendor if needed. Due to the faster upgrade cycle of SaaS software and small deployment time, the customers need not spend time and money in upgrades and deployment of the software. On the vendors' side, not only the vendors could tap newer markets through SaaS [3], the vendors found consistent periodic revenue from subscription model of SaaS very attractive.

1.2 SaaS: A new Business Model

Today SaaS has become an acceptable mainstream business model for not only the end users but also for enterprises [1]. The main contribution of SaaS is the fact that it has brought radical shifts in the previously existing business models of selling software [9], [4], [7]. SaaS represents a major challenge to the conventional licensed software business model. Traditionally, software was delivered primarily by licensing "shrink-wrapped" products sold through retail channels or arrangements with hardware vendors.

In SaaS model, the software is hosted by the vendor and offered to the customer on subscription. There is no one time up-front fee and customization features are inbuilt in the SaaS software. The user also does not need to spend money on infrastructure as the software is hosted by the vendor. The on demand of SaaS business model means customers avoid expensive, time consuming and laborious upgrade processes and the accompanying investments in training specialized personnel. The ease with which customers can configure a solution that meets their requirements without the time and expense of writing custom code shortens deployment time and accelerates subscription and services revenue.

1.3 Technology enablers for SaaS

SaaS has established itself as a successful method to deliver business services across the Web. [13] lists the market drivers for SaaS which arise mainly due to the economic benefits of SaaS . The market drivers for SaaS have existed over 9 years. The cover feature published by IEEE computer society in October 2003, “Turning software into a service” by Mark Turner et al [15], points that there were significant gaps in technical aspects of delivering software as a service at that time. The market for SaaS solutions has matured only after 2005-2006 due to the emergence and availability of new technologies like web2.0, rich internet application (RIA), SOA, cloud computing, virtualization, etc. . The key enablers of SaaS are

- **Web 2.0** refers to business, social and technology evolutions. Socially this has led to the internet establishing itself as the communication platform per se and the browser becoming the universal user interface. From business point of view, the advent of Web 2.0 in 2005 led to a new wave of web business innovation and had a major impact of Web-centric business process management on dispersed manufacturing and procurement activities, new marketing models and globalized micro-businesses linked via web-centric processes [10]. Technologically the ability to read and write against dynamic, near real-time data and information and to program against remote functionality to create a new class of web applications leveraging those capabilities has enabled SaaS to emerge.
- **Rich Internet applications** (RIAs) are web applications that have the features and functionality of traditional desktop applications [2]. RIAs typically transfer the processing necessary for the user interface to the web client but keep the bulk of the data (i.e., maintaining the state of the program, the data, etc.) back on the application server. RIAs typically run in a web browser, or do not require software installation and run locally in a secure environment called a sandbox [16]. The concepts of SaaS and RIA are tightly linked: the potential success of SaaS naturally relies on the market's potential to produce good web applications: people will not be willing to give up their traditional software unless the web-based alternatives prove to be equally good [17].
- **Service-oriented architecture** (SOA) has established its value in enterprise systems as architecture for integration, consolidation, and reuse [8]. As more services become available over the Web as part of the SaaS trend, the way services are implemented will change such that instead of implementing all logic by themselves or delegating to some local third-party product, companies will delegate to some remote service delivered by some third party viz., SaaS. If software is supplied via SaaS using an SOA approach, companies can employ SOA ESBs between external SaaS applications and internal line-of-business applications.
- **Cloud computing** can further leverage the economic benefits of SaaS for the vendors by not investing in the hosting infrastructure and utilising the cloud services to offer SaaS solutions. The availability of data, software, hardware and IT services anytime everywhere provide new opportunities to SaaS vendors to offer more reliable, fault-tolerant and more economical SaaS solutions. Virtualization is the key behind cloud computing.
- **Virtualization** is an abstraction of computer resources. Wikipedia states that "[It] creates an external interface that hides an underlying implementation." Virtualization is a means to maximize the use of existing hardware, operating system or application resources because it can be used to simulate a resource that is not physically there. This helps SaaS vendors in scaling the SaaS software without incurring the cost of extra servers and

leads to cost saving at vendor's end which passes on to the customers.

1.4 SaaS and configurability

SaaS Software as a service (SaaS) is changing the way software is being delivered to the customers. Never before the customer could subscribe to software that he/she could configure to suit his/her needs. SaaS is hosted software which is distributed via internet. This is an economic proposition which takes advantage of the fact that the total cost of a service goes down due to economy of scales. An overview of the software-as-a-service (SaaS) model for software delivery and a high-level description of the architecture of a SaaS application were first described in a white paper published by Microsoft [3]. The success of SaaS depends greatly on how the SaaS software is built to support this new software delivery model besides many other factors like market conditions, consumers' trust, etc. SaaS applications are single instance multi-tenant models. Multi-tenancy does not allow code to be customized. . It is impossible for the vendor to customize the single instance software for any of the customers as the code then will be customized for all the other customers as well. Configurability in SaaS aims to provide tenants/customers with a multitude of options and variations using a single code base, such that it is possible for each tenant to have a unique software configuration.

Configurability is the key foundation stone of any SaaS software. Without it SaaS will be essentially a limited ASP. In this paper we focus on this issue. Our basic objective is to understand the nature of configurability, how it can be provided and the technologies needed to support configurability. In the next section we discuss why configurability is required in SaaS software and what could be the scope of configurability and what features a SaaS software should have to support configurability. In section 3 we discuss how the major different types of configurability can be supported. Section 4 discuses the proposed architecture and section 5 presents implementation of the case study undertaken for building a sample SaaS software. Finally there is the conclusion and future work and the references have been listed.

2. CONFIGURABILITY IN SAAS

The single instance multiple tenants model of SaaS software requires that every SaaS vendor provide support for configurability. Though most of the organizations today have standard software requirements like CRM, ERP etc., each of these organizations have some unique needs regarding user interface, data, business processes and rules, etc. Configurability in SaaS aims to fulfill most of these unique needs through a common web interface to be used by the designer (a privileged user of the organization) of the SaaS application. Configurability allows a unique user experience to each customer of the SaaS application even though the code base is same. The configurability should be such that it is easy and intuitive for the designers and satisfies the needs of the players. The SaaS software's architecture should support configurability without which it would be impossible to use the single instance of the software for different customers.

2.1 Designer and user at tenants

Configurability to support Multi-tenancy leads to the requirement that there exists configuration data for each tenant. The configuration data is created and used by each tenant. Not all users in the tenant could be allowed to create configuration data for the tenant. This leads to the fact that there need to be two different kinds of users of any SaaS application – designer and user. The designer configures the SaaS application which essentially means setting up some config (configurability) data for the users in his organization. When the application is used by the users in the organization, this config data is retrieved by the SaaS software to provide the customized SaaS application to the users.

To cater to these two different kinds of users, the SaaS application has two kinds of interfaces – a designer interface and the user interface. The designer interface is used by the tenant's designer to configure (edit) the SaaS application and the user interface is used by the tenant's users to run (play) the configured SaaS application. This leads to two aspects of any SaaS application architecture – the designer's aspect and the user's aspect. From the designer's perspective, the SaaS application architecture should support capturing config data. From the user's perspective the SaaS application architecture should support parsing the config data and running the configured SaaS application.

2.2 Configuration data and application data

From the discussion in the earlier section, it follows that designer creates the config data and the user consumes the config data for each SaaS application. The config data plays the most important part in realizing configurability in SaaS application. Using the same code base and different config data, the SaaS software is able provide the customized app to the users of different tenant. Other than config data there is application specific data which is available to the users while running the SaaS application. The application specific (app) data is separate from config data. In any SaaS application one need to store both app data and config data such that they can be used effectively and efficiently.

2.3 Configurable aspects of SaaS software

There can be different aspects of SaaS software which can be configured. The following sub-sections explore configurability in different aspects of SaaS software.

2.3.1 User Interface

Configurability of user interface means the ability to change the look and feel of the UI available to the players. A customer can configure the user interface to change its look and feel or to reflect corporate branding. The UI features like icons, colors, fonts, titles, etc. can be changed. This should cater for not only adding/editing/deleting different controls on the forms but also add/delete/edit forms in the user interface. Depending on the type of control each control needs to be configured for placement on the form, a unique ID in the form, data binding and event attachment. There could be option to change the style of the control using skins. To allow for data binding there needs to be option for adding variables. The events could be predefined or could be custom made combining the already existing events as a workflow.

2.3.2 Workflow

Other than configuring the UI to give unique look and feel to the SaaS applications, it is important that the designer be able to configure the behavior of the application. This is important because it is possible that the same kind of workflow may have different behaviors in different organization. A workflow consists of a set of activities, roles and rules and the configuration of these define the behavior of the SaaS application. Workflows allow automation of processes involving human and machine-based activities. Since SaaS enabled software are good choices for applications where web based collaborations are needed, having support for workflow management is a must. Workflows can be either for collaboration among different users or to compose bigger operations out of the existing operations. A facility to build the meta-model of the workflow needs to be available in the designer part of the SaaS software. This is similar to configuring UI.

2.3.3 Data

Data is at the heart of any SaaS application. Data drives the SaaS applications. In case of a particular type of SaaS application, that is, in a specific domain like banking, there will be similar type of data across different tenants. But there will be some unique features in the database of each tenant. So it is possible to provide a template for storing data which meets the most common requirements of the tenants with an option to add the tenant's specific data requirements like adding addition fields to a table or adding additional table or constraints. Thus data Configurability is needed to cater for extensibility in the data model as required by the tenants.

2.3.4 Access control

Each tenant using the SaaS software will have multiple individuals using the software. The responsibility for creating individual accounts for end users, and for determining which resources and functions each user should be allowed to access lies with the tenant. Since there will always be an organization's specific access control data, it is necessary that the designer is able to create, edit or delete roles/users specific to his organization. Users will be grouped into different roles according to the organizational structure and the access control privileges can be configured for each of these. The access control privileges specify what data and UI (forms) an entity in a particular role can access. There might be some data which can only be viewed (only read permission) while there might be some other data which is editable (read and write permission) by users of a particular role. There still needs to be some user configuration to be done as each user should have a unique user id and password. Hence the designer module must cater for building both roles and users to be able to configure access control for a tenant.

While catering for configurability, the designer module for any SaaS software must be able to deal with multiple levels/scopes of configuration units which might be required by the tenants. Since any organization is generally organized into a hierarchical structure, these scopes are also hierarchical in nature. The idea behind the scopes is that the users in top scope have the superset of all configuration options available to the tenant. The users in the lower levels/scopes have a set of configuration options available to the tenant. Whether child nodes inherit and override configuration settings from parent nodes has to be defined in a relationship strategy of the scopes. Configurability options should be available to each tenant to set the scopes as required by the tenant.

2.3.5 Other configurability concerns

There can be more configurability options like domain specific extensions in case of SaaS software in the horizontals like customer relationship management (CRM) in case of tenants in diverse domains like IT services, insurance, etc. There could be configurability options for multiple locales for setting up language, date, time, etc which could be a requirement of tenants whose offices are geographically distributed across the globe. What and how much configurability to provide in the SaaS software depends on the targeted customers' needs.

Since SaaS software is configured by the users themselves, due importance should be given to designing the configuration interface along with designing the interface for end users. Configuring the application should be simple and intuitive. The wizard/screens should present all available options without causing information overload. It should be clear to the user as what options can and cannot be changed within a given scope. The user experience while configuring a SaaS application plays an important role in success/acceptance of any SaaS software.

3. CASE STUDY

3.1 Problem definition

The sample problem taken for building SaaS application is University Grading Management System. There exists a university grading system in all the universities. All universities grade their students. The faculty inputs the grade for the courses they are teaching for the students who take the course. The result is made and the students are able to view the grades. There might be other people like course-coordinator, registrar, etc., who are also involved in the overall process of grading. Let us take two universities like IIT Kanpur and IIT Delhi. Both have similar grading management requirements but there are some differences as listed in Table 1. Thus it makes a good sense to opt for a SaaS enabled university grading management system as software to provide a single solution to both the universities.

Table 1. Some of the differences in the university grading management systems of IIT Delhi and IIT Kanpur.

IIT Delhi	IIT Kanpur
UI –The background colour is blue and the logo is iitdelhi.jpg.	UI – The background colour is beige and the logo is iitkanpur.jpg.
Workflow – If total no. of credits for a student fall not greater than 15 credits short of fulfilling the degree requirements in Mtech. Course, DIIT degree is granted.	Workflow – If total no. of credits for a student fall short of fulfilling the degree requirements in Mtech. Course, degree is refused.
Data –Courses, faculty, grading weightages and grades are different from IIT Kanpur.	Data –Courses, faculty, grading weightages and grades are different from IIT Delhi.
Access control – There are four roles viz., faculty, students, registrar, course-coordinator. The users are different from those in IIT Kanpur.	Access control – There are three roles viz., faculty, students, registrar and the users are different from those in IIT Delhi.

3.2 Solution architecture

The University Grading management System is a web based software for automating the universities' grading system as depicted in Figure 1. As pointed out earlier, this software is also used by two kinds of users – designer and user. The designer uses the software in edit mode and the user uses the software in the play mode. In the edit mode the designer configures the application as required by a specific university. In the edit mode the user (can be student, professor, registrar, etc.) uses the application already configured by the designer.

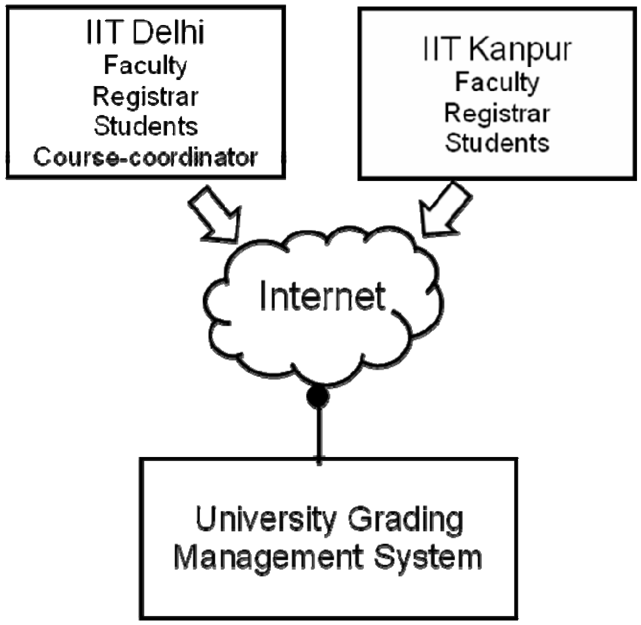


Figure 1: SaaS enabled university grading mgmt system

Keeping in mind, the two different types of users of any SaaS application, the SaaS application architecture can be perceived as in Figure 2. There are 4 key components of the architecture.

1. **Template data:**
   It is impossible to build generalized SaaS software to fulfill needs of all kinds of customers as such an application will have a very complex configurability feature. This will increase the difficulty level in configuring the application as well developing the application and also will increase the vulnerabilities in the software. For example, if we give the designer the flexibility of uploading a code fragment in order to create a new UI element or data elements or activity in the workflow, the software may become vulnerable to say SQL injection. Thus a set of templates (template data) is provided to the designers to help in configuring the SaaS application which contain almost all possible features in a domain. The catch here is therefore to be able to model the domain universe correctly and completely.
2. **Config. Data:**
   In config data module, config data file/files exist specifying the configuration settings for each user's data.
3. **App. Data:**
   In this module, the organization specific application data exist for each customer of the SaaS application.
4. **Application module:**
   This consists of two sub-modules:
   a) Designer module
   The designer module provides the interface to the designer to set the config data. The designer accesses the template data and sets the config data for configuring user interface, workflow, data and access control for the users in his organization through this module. The designer also sets some of the app. Data.
   b) User module
   When the user logs in to the user module, it retrieves the config data specific to each individual user from the config data module and sets the user specific configuration for the SaaS application. The user also retrieves the app data set by the designer and sets his app data.

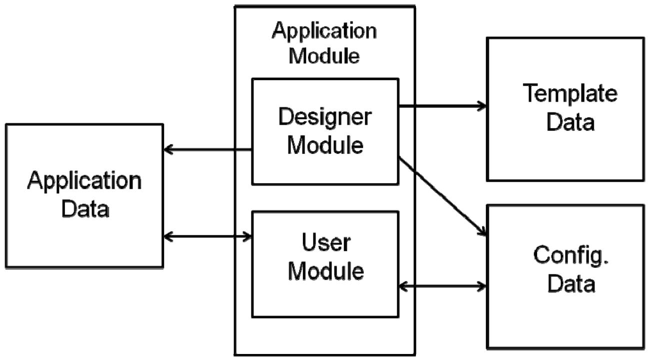


Figure 2: SaaS application architecture

Further going inside the details of the application module, we see that the designer module consists of SaaS application set up wizard and the user module is actually the SaaS runtime engine. The SaaS runtime engine is the backbone of the SaaS application. It loads the required config data to give a unique user experience to each user. Template data is a set of xml files which is used by the designer to create the config data, which is another set of xml files, corresponding to each user. There also exists a set of operations in the application module or the template data (as script files) which are a set of event handlers/functions defined in a language like java, C#, VB or jscript in a class/file.

The SaaS application architecture can be perceived as having two different views– the user view and the designer view. In the user's view there is an Application class which starts the SaaS application and depending on the config data for UI, workflow, access control and data and the connected activities/operations provides unique look & feel to the application. It launches the other interface classes to interact with the app data and the different types of config data.

The user's view of SaaS architecture is as shown in Figure 3. The user invokes the Application class which invokes the access control class and gets a login screen. The user is validated using the access control config data of the tenant. After the user successfully logs in, the control transfers to user interface class which retrieves the user specific UI config data to provide customized look and feel to the application. The workflow engine class is invoked which retrieves the tenant specific workflow config data to provide customized workflow to the application. There is a different database for each tenant. The database interface class uses the database config data to interact with the tenant's database.

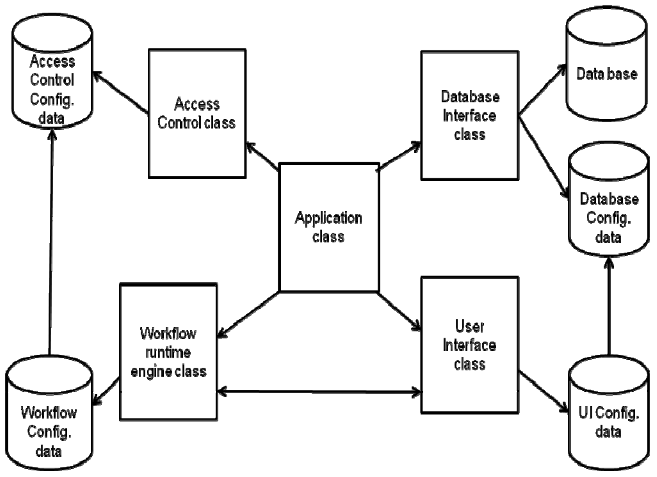


Figure 3: Application architecture from user's point of view

The designer's view involves the use of a configurability interface class and also the templates. Here the application class launches an interactive Configurability Interface class to

1. Configure the database schema using provided templates and store the database config data in an XML file.
2. Configure the access control using the predefined roles in the access control class, create users (logins and passwords) and store the access control config data in an XAML file.
3. Configure the user's user interface using the predefined event handlers and store the UI config data in an XAML file.
4. Design the workflow either through the configurability interface class or through launching the workflow runtime engine and store the workflow config data in an XAML file.

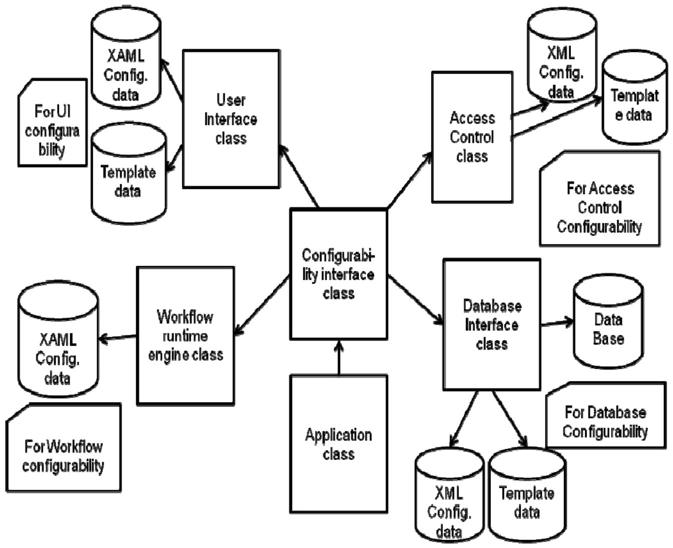


Figure 4: Application architecture from designer's point of view

The designer's view of SaaS architecture is as shown in Figure 4. The proposed SaaS architecture have been conceived keeping in view that it should be such that there is loose coupling between the application module, app. data, config data and template data such that

a. The four can be mixed and matched to provide highly configurable applications.

b. Updating/modification in any one of them do not affect others.

c. Code base remains same across all the tenants.

4. SUPPORTING CONFIGURABILITY

4.1 Data storage to support configurability

Representation of config data is an important consideration when designing for configurability in SaaS application. A desirable feature in the format of config data would be extensibility. One of the ways to represent config data could be using xml which allows for extensibility. All types of config data could be captured and represented using xml markups to cater for configurability. Another possibility of storing config data could be any relational database storage. Before deciding on any kind of storage strategy for config data and app data, one needs to make a comparison between xml and relational databases which represent and structure data in very different ways [5].

XML and relational databases are two different technology sets which are very different from each other. XML supports hierarchical data model, whereas databases support relational data model. Relational databases support page-level or row-level record locking to preserve the integrity of data during concurrent usage and control access to data while it is being updated. There is no in built mechanism in XML to control access to data. It has to be taken care of as file I/O by the application through XML parser. Relational databases provide transaction and rollback support which is not present in case of XML.

There are two requirements for SaaS software wrt. data storage - one is for storing config data and the other is for storing app data. Unlike app data which follows normally relational data model used in an enterprise, config data is generally hierarchal in nature. Let us see UI config data. UI config data has root node as UI which has got many forms and each form has many controls and each control has many properties. Similarly the workflow config data has app as the root node. Each app has many workflows and each workflow consists of activities and/or rules and/or other sub- workflows. The access control data is also hierarchical in the sense that each tenant may have one or more apps. Each app has one or more roles and each role has one or more users.

The other points to consider when deciding on the storage of config data and app data is that unlike app data which is generally huge for enterprises, config data is small in size. Also app data keeps getting updated whereas config data is hardly ever updated. So the config data for UI, workflow and access control are good candidates for using XML. Config data for app data (which may be relational or hierarchical, but small and less frequently updated like other types of config data) may be stored in XML or RDBMS format or may not be required to be stored explicitly if the RDBMS provides access to the metadata.

In case of app data, it makes more sense to use relational database storage because of the relational nature of data, data security needs of the tenants and the need for concurrent access by the users of a tenant. Also in many cases transaction and rollback support may be needed by the tenants. Unlike XML, RDBMS support all these required features efficiently.

4.2 Supporting configurable aspects of SaaS software

UI Configurability could be supported by allowing for changes in the UI through templates comprising of style sheets and skins. To provide better configurability of user interface, a GUI builder could be provided to the designer. The UI config data could be stored as an xml file or database file. Whenever a specific user of a tenant logs in, the UI config data for the corresponding user is retrieved and the specific UI is displayed to the user.

The workflows are stored in workflow specific config data formats. Whenever a workflow is triggered, its data is fetched and it is uploaded in the specific user's login. The access control config data can be stored in a secured database or xml file. Data is stored in database files mainly because tenants have huge amount of data. The corresponding files have to be retrieved at runtime whenever a user accesses the SaaS application. The data is isolated for different tenants by using one of the schemes for supporting multi-tenancy as described in [6].

5. IMPLEMENTATION OF CASE STUDY

There can be 3 ways in which one can build SaaS software. The first would be using core language like C, C++, Java, etc. The second way would be to use an integrated development environment like Visual Studio or Eclipse. The third way would be to use SaaS development environments, like Salesforce.com, available on the net.

Microsoft's .Net framework was chosen for implementation of the case study University Grading Management System (UGMS) application as it supports web based software development and there is a provision for declarative programming in .Net. The implementation of University grading management system was done using .Net3.0 technologies using Visual Studio 2008. The designer module was developed using Asp.net, C#, Xaml and Linq. The UI config data was stored in Xaml which is a declarative XML-based language created by Microsoft. The access control, data and workflow config data were stored in Xml. Linq was used to interact with the configuration data. The designer module captures the configuration data in Xml/Xaml format. This configuration data is used to drive the player module.

The player module consists of web clients and common operations and workflows at the backend. The web clients were made using Silverlight 2.0 and were hosted inside Asp.Net server page. Microsoft Silverlight is a cross-browser, cross-platform, and cross-device plug-in that can render UI using the UI config data stored in Xaml. The web clients interact with the common backend code base through web services built using Windows communication Foundation (WCF). Config data was kept in XML file and Linq was used for interacting with config data. Application data was kept in database and Ado.net used for dynamically generating and executing queries at runtime. This feature is required as we cannot write separate queries for different customers in the code beforehand and compile them. .Net 3.0 provides a framework for workflow development known as Windows Workflow Foundation (WF) in which one can use XAML format to declaratively implement workflow. Since connectivity between Silverlight and workflow foundation (WF) was not directly supported in Silverlight 2.0 at the time of implementation, own XML file format was used for storing and executing workflows. Though integrating WF and Silverlight was attempted as the both are technologies in .Net framework. A run-time view of SaaS application developed using .Net is shown in figure 5.

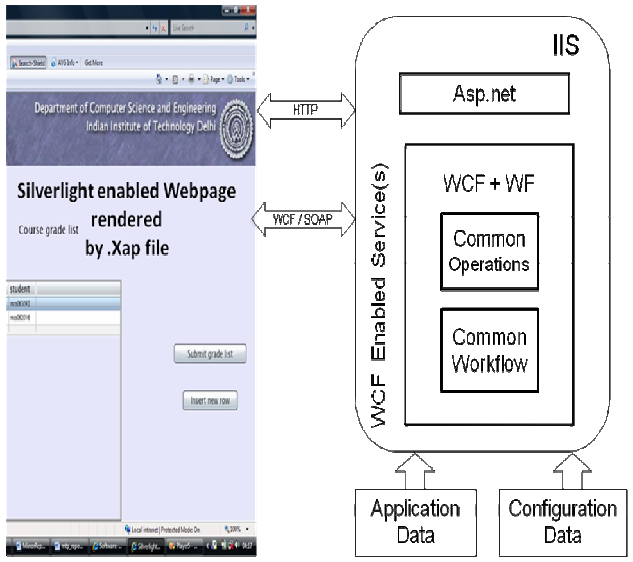
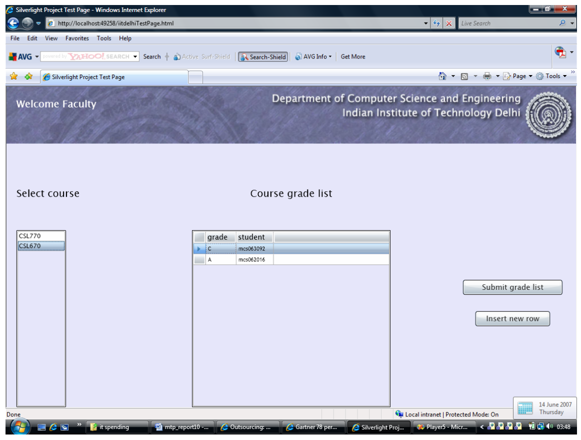


Figure 5: Run time view of SaaS application using .Net technologies

When a user of either IIT Delhi or IIT Kanpur logs in play mode, the player module is instantiated. The player module fetches the config data corresponding to the user and displays the proper UI and executes the workflow as configured by the designer earlier. When one of the IIT Delhi faculty logs in to the player mode, a screen as shown in Figure 6 shows up. When an IIT Kanpur faculty logs in, he gets a user interface as shown in Figure 7. Thus though the code base is same, the difference in the user interfaces is due to the UI configuration data created by the respective designers of the two universities. This stored config data is retrieved by the code to give different look and feel for the faculty of the respective universities.


Silverlight Project Test Page - Windows Internet Explorer
http://localhost:49258/iitdelhiTestPage.html
File Edit View Favorites Tools Help
Silverlight Project Test Page
Submit grade list
Insert new row
Done
Local intranet | Protected Mode: On

Welcome Faculty

Department of Computer Science and Engineering
Indian Institute of Technology Delhi

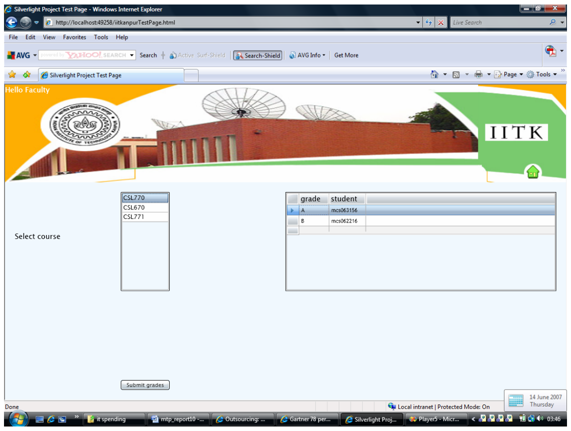
Select course: CSL770, CSL670

Course grade list

grade	student
C	mcs063092
A	mcs062016

Figure 6: IIT Delhi faculty page

The same holds true for access control, workflow and data. The respective config data corresponding to each tenant is retrieved and used to give each tenant an application with different access control and workflow behavior and also different data extensions to the respective data.



grade	student
A	mcs063156
B	mcs062216

Figure 7: IIT Kanpur faculty page

6. CONCLUSION AND FUTURE WORK

Configurability in SaaS enables customers to implement a system which can be used in a very short amount of time, bypassing on premise challenges such as server provisioning and software installation. However SaaS does not mean only being able to do configuration. Salesforce.com as an example provides a very flexible platform for building one's own custom objects and UI, as do many other vendors. This enables customers to model their business process which is unique to them. The keys to "infinitely configurability" are standards and decomposed services. The standards provide automated service-oriented ways in which composite solutions come together with less work, and decomposed services ensure there is useful content to draw from.

Configurability allows for some unique features in each tenant's application. But a tenant should not expect everything unique in the application. If one needs everything customized, one won't have success with SaaS. SaaS makes sense if the process is simple enough to be configured. SaaS is a revolutionary approach to using software which is beneficial to both customers and vendors. The key requirement being, it is architected correctly. SaaS requires an architecture that supports end user configuration. The emerging technologies in .Net framework enable building SaaS applications.

SaaS is one of the biggest technology trends to affect business applications in recent years. Due to evolving market place and emerging technologies, SaaS architecture details and implementation details are open to research. Readers can refer [12], which gives a comprehensive listing of SaaS related papers published by Gartner, to familiarize about the key SaaS issues. One of the key issues for SaaS in the year 2008 according to [11] is that how will SaaS technology and architecture meet the needs of evolving business models.

7. REFERENCES

[1] Ben Pring, Alexa Bona, James Holincheck, Michele Cantara, Yefim V. Natis, Gartner. 3 January 2008. *Predicts 2008: SaaS Gathers Momentum and Impact.* s.l. : Gartner, 3 January 2008.

[2] Brijesh Deb, Sunil G.Bannur, Shaurabh Bharti. Jan 2007. *Rich Internet Applications (RIAs), Opportunities and challenges for enterprises.* s.l. : Infosys, Jan 2007.

[3] Carraro, Frederick Chong and Gianpaolo. 2006. *Architecture Strategies for Catching the Long Tail.* s.l. : Microsoft Corporation, 2006.

[4] Dodge, Don. 2006. SaaS - new software model, new challenges. [Online] April 12, 2006. http://dondodge.typepad.com/the_next_big_thing/2006/04/saas_software_s.html.

[5] Erl, Thomas. 2004. Service-Oriented Architecture - A field guide to integrating XML and web services. s.l. : Pearson Education, Inc., 2004.

[6] Frederick Chong, Gianpaolo Carraro and Roger Wolter. 2006. *Multi-Tenant Data Architecture.* s.l. : Microsoft Corporation, 2006.

[7] Gregoire, Michael. 2007. Between the Lines: SaaS Business Model Execution from Top to Bottom. [Online] December 03, 2007. **www.sandhill.com/opinion/editorial.php?id=163**.

[8] Houlding, David. 2007. From SOA to SaaS. [Online] Dr. Dobb's, March 02, 2007. www.ddj.com/web-development/197700752.

[9] McPherron, Kate. 2008. Software-as-a-Service (SaaS): The New Paradigm for the Software Industry. [Online] February 10, 2008. www1.sao.org/Resource_Center/newsletter_articles/200802/Kate_McPherron_Software_as_a_Service.php.

[10] Phife, Gene. 2007. *Continue to Drive Business Innovation.* s.l. : Gartner, 2007.

[11] Robert P Desisto, Ben Pring. 21 March 2008. *Key Issues for Software as a Service.* s.l. : Gartner, 21 March 2008.

[12] Robert P. Desisto, Ben Pring. 2008. *Essential SaaS overview and guide to SaaS.* s.l. : Gartner, 2008.

[13] Sharon A. Mertz, Chad Eschinger. 2007. *Dataquest Insight: SaaS Demand Set to Outpace Enterprise Application Software Market.* s.l. : Gartner, 2007.

[14] *Software as a service: strategic backgrounder.* s.l. : Software and Information Industry Association.

[15] *Turning software into a service.* Mark Turner, David Budgen, Pearl Brereton. 2003. X, s.l. : IEEE Computer Society, 2003, Vol. 36, pp. 38-44.[21]

[16] Wendland, Sandra. 2007. How RIA enables "Software as a Service" for software manufacturers. [Online] June 4, 2007. http://canoo.com/blog/2007/06/04/how-ria-enables-saas/.

[17] Wolf, Dave. 2008. The Business Case for Rich Internet Applications. [Online] Jan 31, 2008. http://pbdj.sys-con.com/read/452386_1.htm.