Software Design on a SaaS Platform*

Guoling Liu, He Jiang, Runian Geng
School of Information Science and Technology
Shandong Institute of Light Industry
Jinan, China
maindollar@126.com

***Abstract*—This paper explores the concept of *software as a service*, which envisages a demand-led software market in which businesses assemble and provide services when needed to address a particular requirement. Comparison between SaaS and its counterparts are described. The design considerations and customer life cycle in SaaS is analyzed. Lastly, the implements for industry commercial prototype are given.**

Keywords-SaaS; software design pattern; software architecture; service-based software

I. INTRODUCTION

In 1999 the Pennine Group (http://www.service-oriented.com), a consortium of software engineering researchers from the University of Durham, Keele University and UMIST, put forward the view that the future of software lay not in developing new architectural styles based upon constructional forms, such as objects or components, but in taking a radically different view of the way that its functionality was delivered to the user. This idea, which we termed Software as a Service (SaaS), is one of a demand-led software market in which services are assembled and provided as and when needed to address a particular requirement.

Desktop software and Internet applications are going through changes; the two are merging together. Desktop functionality and richness in interaction is now possible on the web. SaaS establishes a model to build a business on these advances. The SaaS idea of using software on the internet lets you think about a fundamental change in how we use our computers nowadays; the complete integration of the Internet in your operating system. When all software runs on a Web server eventually you will stop needing the noisy, extra-large computer case below your desk. Never again will you encounter the infamous blue-screen of death (that is, for the Windows users among us), because you simply won't have any hardware at home except for your monitor and Internet connection. All you will have to do is log in to your personal online desktop. You will have no software installed; everything runs from a Web server. Automatically all your files and settings are also stored remotely.

This work is partly supported by The Project of Shandong Province Higher Educational Science and Technology Program (No. J09LG07), Shandong Province Natural Science Foundation(No. Y2008G26) and the Project of Youth Science and Technology Star Program of Jinan (No. 20090202).

More concrete; SaaS is a software business model. Software is not offered as a product, but as a service. Software runs on a web server and the user uses it through an Internet connection. The user only pays for using the software instead of for owning it. Today, with advances in infrastructure and software technology, a new breed of application is being produced: robust Web-based applications designed as centralized, shared-instance, multi-tenant applications called software as a service. Like ASPs, up-and-coming SaaS solution provides take responsibility for managing the service including security, performance, availability, reliability, and scalability. However, the similarities stop there. In mature SaaS offerings, multi-tenancy applies to all tiers of the software architecture: all tenants share the same codebase and instances of the application, which enables large economies of scale. Given that the underlying application codebase cannot be changed for each tenant's customizations, the application is abstracted into loosely coupled fine-grained configurable components. This lets customers quickly change the presentation, logic, and database layers of the application through on-screen clicks without code modification, compilation, or deployment.

SaaS is driving a new implementation methodology. With no infrastructure to be purchased, set up, or maintained, a SaaS application is immediately available for customization. A declarative development model enables changes to be quickly designed and implemented. Releases can be smaller and more frequent as a result of logical partitioning of functionalities and the decreased overhead for testing and deployment. SaaS has enabled a shift from traditional waterfall-like development models consisting of single-phrase sequential steps to a more agile methodology consisting of multiple phases and multiple iterations within each phrase. Iterative methodologies help maximize user adoption by incorporating user feedback early in the discovery, design, and development stages of the implementation.

SaaS is rapidly growing. Its market was estimated to grow about 25% per year from 2006 and reach $10 billion in annual revenue by 2009. Well-known SaaS products include Salsforce.com and Employease.com. The content of these products varies, including Customer Relationship Management (CRM) service, Human Resource Management (HRM) service, desktop functionality, email, and supply chain and inventory control [1, 4].

This paper aims to describe the service-related protocols, analyze the considerations and customer life cycle s for SaaS and discuss the implements for service technology

978-1-4244-6349-7/10/$26.00 ©2010 IEEE

II. CONTEXT AND DEFINITION

A. SaaS and Legacy Software

There are three primary differences between SaaS and legacy software: delivering different products, adopting distinct pricing modes, and employing different delivery methods.

- They deliver different products. SaaS provides the subscribers with a collection of standard applications and services. Legacy software provides customized applications to its users.
- They adopt distinct pricing modes. Legacy software users purchase the application. SaaS subscribers pay for each time they use the services.
- They employ different delivery methods. Legacy software applications are installed on users' premise. SaaS applications are located in services providers' central server and the services are delivered via the Internet or an intranet.

The main difference is that with traditional in-house software the user only focuses on the application itself. The application is stand-alone, installed on the users system and the IT department or network administrator takes care of the management, monitoring, updates and all other tasks related to using software in a business environment [2].

A SaaS application also focuses on this primary application part, but incorporates tasks that all together make up the SaaS application and business model. These tasks include registering, paying for the software, using support and customizing the application. A SaaS application is much more specifically directed to the customer instead of a traditional desktop application directed at the mass. This results in a higher value of design and interaction.

For instance, when the registering procedure of a SaaS application is not enough intuitive, fluent or clear, a potential client abandons his efforts instantly. A case like that immediately results in the service provider not making money. Desktop packages, on the contrary, are sold and installed in large numbers. In a business situation a user is more or less obliged to use that particular piece of software when installed. In an at home situation the user is at least more determent to learn how to use the software, because he has paid a certain amount of money for owning the package. A SaaS application does not consist of a single user experience like a traditional desktop application. SaaS combines multiple user experiences and each of those experiences must be designed with the same user interface and procedure.

Furthermore, SaaS applications are accessed over the Internet; therefore users will perceive the use of the application similar to that of a web page. They will transfer their knowledge and previous experiences with websites to the application. However, when a SaaS application is used, we want the user to experience it as the primary active application. To avoid the quick navigation behavior users are familiar with in classic web pages a SaaS application must be used full-screen and requires the user's full attention.

B. SaaS and SOA

Service Oriented Architecture (SOA) is a means of designing and building software. It is a manufacturing model. Software as a Service (SaaS) is a means of receiving software through an external party to your business similar to telephone or power utilities. It is a sales and distribution model. Both SOA and SaaS are hot topics in IT as both deliver in very specific ways material reduction in costs of IT operations and deliver a more agile alignment between business needs and IT solutions. The manner in which those benefits are delivered is the key difference. And in many ways, to capture the majority of the cost reduction and agility benefits it is often best to incorporate both SOA and SaaS in IT landscape.

SOA provides a means to deploy and quickly re-configure as business conditions change the applications and databases hosted in data centers owned by own company. Recent versions of many department and enterprise application stacks from major software providers deliver products with a SOA design and implementation. Often ERP functions such as order management, general ledger, and manufacturing planning are still predominately delivered in this fashion [3].

SaaS provides a readily accessible means to out-source the applications and databases of a business. Some key examples include sales force automation, payroll, expense reporting and recruiting. One of the key considerations with SaaS is how to effectively integrate these outsourced applications with the internally hosted ones.

C. Advantages/Disadvantages of SaaS

Software as a Service is geared towards specific type of business. Although they can easily work in most enterprise settings, there are certain requirements SaaS would have that make it undesirable for some businesses.

- Powerful Internet Connection Required: although connection online is available almost everywhere, the rate of connection is never the same. Some areas can't provide strong internet connection and SaaS (as an online application) will have to load everything in the browser. The expected function might not even move forward without strong internet connectivity.
- Increased Security Risk: attacks are highly likely if everything is launched online. This is probably the most challenging part in SaaS and in Cloud Computing industry. SaaS has increase security concerns compared to other platforms because of its consistent interaction with different users.
- Load Balancing Feature: one of the challenges the business would face in cloud computing and all SaaS applications is load balancing. Although industry giants offer load balancing, it will still require consistent monitoring from businesses [3, 4].

SaaS is getting better and better as new trends in the industry are slowly being implemented. Among the trends in cloud computing is the powerful integration of API or Application Programming Interface. Although SaaS could

provide the functionality the business needs, upgrades are important to keep up with the demands. Instead of changing the application, businesses will just add an API in their application. The integration is easy and maximum efficiency of the additional function is expected.

III. Design Considerations And Life Cycle In SaaS

A. SaaS and ASP

SaaS is another version of the failed Application Service provider, or ASP (application service provider), and other hosting models of the past. According to M. de van der Schueren (Chairman of the Dutch ASP-forum) SaaS is just a marketing term; in fact, ASP and SaaS are basically the same.

The basic principle of hosting applications externally was re-branded because of the previously failed attempts. The important thing is that now the environment seems to be ready for, and reacting positively on, SaaS developments.

ASP and hosting companies in the web 1.0 era failed for two reasons. First, they did not fundamentally change the architecture of their software (like we do with our now available web 2.0 Flash, AJAX and XML techniques, true genius!), but simply sold applications to organizations that did not want to house them on their own systems. The hosting costs proved to be too much for the ASPs to withstand. Second, only a small segment of the market was willing to outsource their application needs. They considered their business applications a strategic asset and preferred to have them under their own roof.

But times have changed; the boundaries are gone. Today's economic and competitive pressure makes nearly any form of outsourcing fair game. The evolution of the Internet and the central place it takes, also in business, make companies more flexible and open to the possibilities of the web. This, together with technical advances, decreased cost of investment and increased reliability, makes SaaS more attractive than ASP and hosting models of the past.

B. SaaS and General Web-based

On an application and interface design level a standard web-based software application may not be very different from a SaaS application, because it works with the same server- and client-side technology and interface behaviors. Essentially a SaaS application is just a standard web-based software application, but (and there is a large but here) with a lot of added SaaS-specific functions. If a web-based software application is empowered by a SaaS business model the application must contain a number of services that enable a company to make money with the application and a client to use it. These include registering, billing, support and the other examples I mentioned above [6].

With SaaS the user interface is much more important than with a standard or free web-based software application, because a companies business depends on it. A user is free to choose for your SaaS application or not, and even when he does he can just as easy quit using without having seriously invested. The customer-driven and customer-focused approach of SaaS obliges a SaaS application to be completely customer friendly. The power lies with your user with SaaS, when you don't serve him you will not make money. It is as simple as that. Therefore, the business of a SaaS provider depends on the user experience of the application as a whole.

Because a SaaS application itself is a web-based software application guidelines can be derived from web-based design guidelines. But they need to be enhanced on points specific for SaaS relating to call-to-action, overall application behavior, and interface consistency in all services next to the application itself.

C. Design Consideration for SaaS

This section provides a summary of factors that a business should consider during the design or adoption of a SaaS solution. These considerations focus on SaaS and its specific properties. These considerations quote Paul Giurata's paper: "Application strategy and design for a profitable SaaS" [7].

- Purchasing/Sign-up
- Provisioning
- Adoption and Penetration
- concurrent multi-tenant high-performance
- mass data storage and access
- customized and convenient user experience
- Seamless opt-in upgrades
- Open API for integration
- Enable application customization
- Open the application full-screen size
- Online training and guidance

D. Customer Life Cycle for SaaS

In traditional enterprise applications, most phases in the customer life cycle are managed by people and services outside of the "core" application. In SaaS these phases are all handled as part of the application. This is the premise behind the SaaS Customer Life Cycle Framework we are developing at Catalyst Resources [8].

With traditional enterprise software, many departments and services are involved in supporting the customer life cycle for a particular client. There is a large upfront sales effort with a protracted sales cycle, from a large team (pre-sales, account manager, marketing) that "manages" the buyer through the sales process. Installation/setup including customization and provisioning requires an in-house IT department or consultant. Training and a post sales support staff manage the usage and on-going maintenance phases.

So with traditional enterprise applications, most of touch points in the customer life cycle are managed by people and services outside of the "core" application (e.g. the CRM or ERP). Therefore the strategy and design of enterprise software is almost exclusively targeted on the usage phase of the life cycle.

In SaaS, all aspects of the customer life cycle are exposed to the customer and all need to be handled as part of the strategy and design of the SaaS. If the SaaS application does not provide a satisfactory experience at each touch point along the way (e.g. acquisition, usage, support, monitoring,

etc), you would get customer abandonment. If the way you handle each touch point does not scale without adding staff, you would lose economies of scale and profitability becomes elusive.

IV. THE IMPLEMENTS FOR SERVICE TECHNOLOGY LAYER

We adopted industry commercial developments as our first prototype development.

- Database: we set up four distributed databases (two IBM DB2 and two MySQL) in four sites using real medical sale data.
- Web services development tools: IBM Application Developer 5.1 was used to develop the services.
- Application Server: IBM Web Sphere application server 5.0 was employed to host the services.
- Service registry: A relational database was used to simulate the service registry.
- Message style: RPC style was used rather than document style.
- Language: Java 1.3 and J2EE were used.

Our development was carried out as follows.

a) Select service protocols

b) Select data storges

c) Select security

d) Selcet various service

e) Service description:

f) Service definition

g) Service registry

h) Service delivery

i) Service composition

V. CONCLUSIONS

SaaS is defined by not only delivery via the Internet, but by subscription and periodic payment. Especially these properties distinguish SaaS from standard web-based software. As stated, a SaaS user experience resolves around a constellation of services, all equally important in establishing a positive user experience, a successful SaaS application, a loyal customer base and, essentially, revenue. This lays the foundation for designing and developing a web-based software application delivered as a service.

In this paper, we analyze the design considerations and customer life cycle in SaaS. At last, we give the contents of industry commercial prototype developments.

REFERENCES

[1] Hai, Henry, Sakoda, Seitaro, “SaaS and integration best practices", Fujitsu Scientific and Technical Journal, v 45, n 3, p. 257-264, July 2009

[2] Manish Godse, Shrikant Mulik, "An Approach for Selecting Software-as-a-Service (SaaS) Product," cloud, pp.155-158, 2009 IEEE International Conference on Cloud Computing, 2009

[3] Thomas Erl, SOA Design Patterns, Prentice Hall PTR, Upper Saddle River, NJ, 2009

[4] Brian Moore, Qusay H. Mahmoud, "A service broker and business model for saas applications," aiccsa, pp.322-329, 2009 IEEE/ACS International Conference on Computer Systems and Applications, 200

[5] Reena Mathew, Ryan Spraetz, "Test Automation on a SaaS Platform," icst, pp.317-325, 2009 International Conference on Software Testing Verification and Validation, 2009

[6] Mark Turner, David Budgen, Pearl Brereton, "Turning Software into a Service," *Computer*, vol. 36, no. 10, pp. 38-44, Oct. 2003, doi:10.1109/MC.2003.1236470

[7] M.T. Hoogvliet, “SaaS Interface Design-Designing web-based software for business purposes,” unpublished.

[8] Paul Giurata, “How to Use Customer Life Cycle as a Strategy for SaaS,” unpublished.