SaaS (Software as a Service) – Models and Infra-Structures

Thesis’s Extended Summary 08/09

João Samuel Nunes Lopes, nº 53917

Supervisor: Prof. Alberto Silva, Accompanist supervisor: Prof. João Saraiva

MEIC, Alameda, Instituto Superior Técnico, Lisboa, Portugal

Abstract: SaaS (Software as a Service), frequent and incorrectly referred to as the ASP (Application Service Provider) model, is considered by many as the new revolution in application software distribution. In par with the Internet’s evolution nowadays, many believe that traditional packaged applications might soon become obsolete in comparison to web-based, outsourced products and services that remove the responsibility for installation, maintenance and upgrades from already exhausted IT staff. While such drastic predictions have not yet happened, the essence of this change – the delivery, management and payment of software as a service rather than a product – is affecting all participants in the software industry. This project will study and compare the traditional, ASP and SaaS models; as well as implement the SaaS model to over the WebComfort platform.

Keywords: Business; Services; IT Outsourcing; Software Distribution Models; Internet; Added Value

1. Introduction

Since its beginning, Information Systems have revolutionized the business world, transforming in great part the way enterprises do business every day. For a long period, the first Information Systems were projected to be installed and run on self contained and autonomous platforms, independent from other systems. As a first and accepted approach to market these systems, software was distributed in a product-based licensing manner. In time, with the evolution of ICT technologies, new platforms arose that allowed for automatic information exchange between various systems. Among other things, this allowed for the development of new kinds of software architectures (i.e. client-server, multi-tier architectures), that allowed even further optimization of business efficiency. As such, many organizations adopted these new technologies, building enterprise networks to broadcast information quickly and effectively between various workstations and implementing enterprise Information Systems to harness these technologies. While larger enterprises were able to develop, deploy and maintain their personal Information Systems, smaller enterprises struggled to do so on their own. As the costs grew, it became nearly impossible for smaller businesses to afford to purchase, deploy and maintain these solutions [1].

Due to this barrier, the Service Provider family of models – of which is part the Application Service Provider (ASP) business model – emerged [2], providing customers with outsourced IT Services. The ASP model in particular supplied its customer’s needs for outsourc-provisioned computer-based services,
delivering these over a network. This was especially appealing for smaller enterprises, since these lacked the high-cost infra-structure necessary to run such systems, in addition to the specialized personnel to carry out its maintenance and upgrades. It offered more flexibility to its customers and was generally cheaper than alternative Legacy models, so the ASP made using the software possible and at the same time eliminated many of the typical related annoyances. As such, since its appearance and especially during the 1990s [5, 6], many organizations adopted this model, achieving new business opportunities and saving time and money to focus on their core competencies. Nonetheless, as it is studied in this work, the ASP model in its original form also had its disadvantages and threats, which led to the bankruptcy of many ASP providers (ASPs) [5]. Despite these setbacks, several ASP providers have managed to survive. Most have either narrowed their focus on a particular market solution, or have adopted other software distribution business models (i.e. SaaS model) [1].

Following the decline of the ASP model, the Software as a Service (SaaS) business model appeared as an ASP successor, to amend its disadvantages and covet its outstanding opportunities [7]. Common but incorrectly referred to as the ASP model, SaaS is in many ways similar to its forerunner. In fact, SaaS can be viewed as a revision to the ASP model, with a clearer business model and a leaner business scope. Like the ASP model, SaaS delivers outsourced computer-based services to its customers via a network, but in this case the network is always the Internet; and applications are developed to be delivered massively to as many customers as possible while achieving maximum instance efficiency (i.e. multi-tenancy technologies).

The evolution of web technologies like JavaScript, CSS or SVG, and the conception of new web standards have recently led to the development of new Internet-delivered web-application concepts like the Web 2.0 and Cloud Computing. Following this, new and powerful frameworks have been developed, to allow to easily harness the Internet as a platform rather than a simple channel [1]. Due to the wide propagation of the Internet, this channel has brought various advantages to the software distribution industry, allowing among other things, for Software Manufacturers (SM) as well as providers to reach their customers quicker and more efficiently. Furthermore, given the technological advances during the last years, the Internet has become a cheap and viable channel for information exchange between various Information Systems and as a consequence it has eliminated the customer’s need for a personal network. What’s more, recently SaaS platforms like Google Apps, Microsoft Windows Azure, Force.com, or Wolf Frameworks, have started to be deployed on the Internet in the form of Platforms as a Service (PaaS). These services directly implement some of the most difficult technological challenges for SaaS developers (i.e. multi-tenancy, scalability, database services on the cloud), providing more familiar, transparent and easy to use APIs to develop new SaaS readied applications. On the other hand, these PaaS solutions also feature access to a mesh of services in the Cloud, which allows to rapidly build solutions based on external services (Cloud Computing).

More and more, applications nowadays are migrating from traditional stand-alone packaged software to online web-based services. Following the idea that “the Internet changes everything”, many believe that Cloud Computing, and subsequently the SaaS business model, will eventually eclipse the traditional packaged software distribution models. This study compares three different groups of software distribution and business models (Legacy, ASP and SaaS), and analyses if these predictions might be truthful, or if the SaaS model might just be the hype of the moment. As a result of this analysis, a discussion between the three model groups is made to assess when and in what circumstances the SaaS model is justified, considering the Traditional and the ASP alternative models. In addition to this, a SaaS model infrastructure applied to the WebComfort framework is proposed and implemented to test the SaaS model in this framework.

2. Problems

In today’s exceedingly competitive, 24x7 business world, IT must deliver applications to workers all around the globe, at all the times. These users work on varied and heterogeneous devices, and access a variety of application solutions. IT must comply with critical business requirements such as regulatory agreements, data security, IT budget cuts, as well as power consumption reductions in order to meet new environmental requirements.

In spite of this, what is observed in many organizations is that IT teams don’t have the infra-structure, knowledge, time and/or funds to support their user’s necessities, and are overwhelmed with just maintaining the IT. Some of the problems these organizations generally encounter when creating and managing their own IT teams include:
### Business Issues
- High entry-cost and expensive IT maintenance budgets;
- IT management might not be directly related with the organization’s competencies, so there is a longer learning curve and economies of scale cannot be applied effectively;
- The quality of the final system solution isn’t generally great and many times are faulty;
- IT teams must be ready to react at any time to invariable and variable problems like giving support to a user or repairing a malfunctioned device;

### Technical Issues
- When installing new applications, IT staff must test, find and resolve all conflicts with existing systems and endpoints (i.e. servers, clients, and machine combinations);
- Each application has to be installed and configured on each endpoint device;
- Application updates take time to be developed or acquired and then installed;
- When an application is updated, testing must be done again;
- Application fixes need to be installed on each endpoint device;
- When an application is to be removed, it must be uninstalled from each endpoint device along with all its former settings, which sometimes can subsist and originate further compatibility conflicts in time;

Overall, what is observed in various cases is that the process of maintaining these systems continuously and up-to-date is expensive, so businesses tend to reduce IT costs and focus on their core competencies. While this can be effective to a level if a business’s IT systems are small or aren’t habitually updated, in enterprises with higher IT requirements this strategy isn’t viable, leading to inaccessible applications, loss of productivity and unhappy customers.

Fortunately, through time, several technological advances (i.e. virtualization and application virtualization) have led to the development of alternatives to the traditional software deployment models, proposing different business, licensing and distribution approaches like the ASP and SaaS models. Nonetheless, these two business models still have setbacks. In a constantly changing segment like the software industry, it is crucial for an organization to understand these different approaches and decide, based on the pros and cons of each model, which is better for it.

### 3. Objectives

The main goal of this project is to analyze and discuss the traditional, ASP and SaaS models and propose and implement an infrastructure capable of reproducing the SaaS model over the WebComfort platform.

Throughout this work, an analysis of the State Of The Art on the three studied groups of business models is made. Concluding the analysis of the State Of The Art, a discussion between the three studied groups of models is made to evaluate when and in what circumstances the SaaS model is justified considering the Legacy and ASP alternative software distribution models.

On the other hand, as a more practical work of this thesis, the main goal of this project is to implement the infrastructure of the SaaS model over the WebComfort platform. Finally, as a result of this work, a working example of a SaaS service provider with diverse services is to be produced, installed and evaluated on an existing and operating WebComfort platform.

### 4. State Of The Art

Ever since the emergence of software solutions, the techniques, processes and methods of software development have been dominated by supply-side issues, giving rise to a software industry oriented towards developers rather than customers. To achieve the levels of functionality, flexibility and time-to-market (TTM) required by customers, a radical shift is required in the development of software, with a more “customer demand” point of view. In the last few years, technological advances have allowed for the increase of new software development and deployment models based on application virtualization. Today, what is observed is that increasingly more organizations are adopting this approach and traditional software vendors are progressively more switching to service oriented approaches. Following is summarized a study of these models pondering the pros and cons of each over specific aspects enunciated in the “Objectives” section of this work.
4.1. Legacy models

Traditionally, Microsoft and almost every other SM, deliver software primarily by licensing "box-wrapped", desktop-based products sold throughout various retail channels, or through agreements with hardware vendors. Since the emergence of these models, software solutions have been deployed massively to various equipment devices and run in local environments – independent from other platforms. However, given the latest technological advances, these business models are being surrogated by new approaches based on application virtualization. As such, throughout this analysis they’ll be grouped as the Legacy models.

Legacy applications are implemented over self contained systems, which don't depend on hosting services and can extensively explore a platform’s resources. Because of this, execution and interaction with these types of applications is very user-friendly. Also because of this, Legacy applications have independency on the network. Applications can access the network, but if a connection to the Network fails, the application can still respond adequately because it is running locally. Also, another pros of this set of models is that it can restrict access to sensitive data. On the other hand, in terms of integration, this model is hardly scalable and very difficult to maintain. What’s more, licensing in this model requires that each application copy has an associated license. This makes buying and maintaining software extremely expensive as bigger as the enterprise is. In general, this model is a good solution for the applications that are used on a single-user basis, and when the application isn’t required to be updated very often. In this case, a customer buys a license, and generally can opt to not contract any support or maintenance.

4.2. ASP model

The Application Service Provider (ASP) model is a software business, licensing and distribution model based on the delivery of computer-based services to customers over a network. It derives from technological advances of the Information and Communication Technologies (ICT). Moreover, the emergence of the first computer networks allowed for the development of new hardware and software structural designs, namely the virtualization and application virtualization that represent the main foundations of this architectural solution.

At the core of the ASP phenomenon is the intricate task of customers to evaluate the cost and risk associated with the acquirement and maintenance of hardware, software, and personnel. The ASP model seeks to correct these issues and provide customer-coveted, new flexibility and efficiency opportunities by providing complete IT solutions at a regular fee payment model.

A great advantage of the ASP model is that each application instance has only to be installed or deployed once, and that each client has to be installed only once (if necessary). Moreover, ASP’s dynamicty in providing solutions that can use both thin-clients (i.e. browsers or custom) or fat-clients. This makes possible to develop very user-friendly interfaces as on the Legacy group of models.

On the other hand, a big disadvantage of this model is its dependence of a network connection. Nonetheless, it can be deployed in private networks, so that makes network dependency less problematic. What’s more, implementing local private networks or VPNs can secure data in an organization. This kind of solution makes the client dependent of the provider, which can be
unacceptable in some businesses. To correct this issue nonetheless, ASPs provide SLA guarantees. Another problem with this kind of solution is that when accepting ASP service provisioning, the client must generally accept the application as it is serviced by the provider, because providers generally have no license to alter the applications they provide. Also, because providers focus on servicing provisioning services, support for the actual applications installed on ASPs is difficult to respond to.

This solution is good for medium to large enterprises, which can support the costs of ASPs, or that privilege data security.

4.3. SaaS model

After many ASP system solutions became disadvantageous, the SaaS model was developed as its successor to harness the opportunities left by a deficient ASP model. It proposes software business, licensing and distribution approaches and is comparatively similar to the ASP model. Like in the ASP model, SaaS delivers computer-based services to customers over a network, but in a different way specializes in delivering services massively to any user over the Internet.

In studying SaaS, many times people interchangeably misuse the concept ASP to qualify SaaS services. This is not truthful as SaaS is not ASP. SaaS differs from ASP because it does not provide application hosting plans for a defined set of customers; rather it provides a predefined service to as many customers as possible. This allows for software businesses to specialize in a small group of services and adapt to what the customers demand – not the other way around like in most other methodologies.

The major advantage of this model is the fact that no software generally has to be deployed, nor do new instances have to be created to service a new customer. This model of service provisioning implements a very focused scope of available services, therefore making support and configurability accessible, cheap and applicable to every customer’s application. Another great feature of this model is its payment methodologies. In this model, a user is generally given the ability to try a service before subscribing to it, and upon subscription can opt to various payment methods, including pay-as-you-go. This model does not involve selling of licenses for the services that are subscribed. As such, users can generally opt out of a service at any given time. This also makes the provider more competitive, to keep the customers happy with their services.

In contrast, a disadvantage of SaaS is that this model uses exclusively the Internet as its channel to provide its services. This means that when the provider’s Internet connection fails, the application becomes unavailable, which in some cases can be unacceptable. Another problem with this model is that it depends on customer’s browsers to access the service, which can be limiting in many ways. Also, because data cannot be stored locally on the client’s computer safely, it has to be stored on the Internet. Because of this, security issues can arise if the data being used is sensitive. This kind of solution also makes the client dependent of the provider, which can be unacceptable in some businesses. To correct this issue nonetheless, SaaS providers can assure SLA guarantees to customers, with associated penalties when these are not complied with.

This solution is good for any sized enterprises or single-users, but is best exploited by smaller enterprises and single-users, since these lack the infrastructure to run alternative solutions. Moreover, because SaaS service provisioning does not lock its customers to the service, users can opt out of the service at any time.

5. Solution

Until now, the WebComfort platform only supported the hosting of WebComfort Applications, over instances that were manually deployed and maintained. Application Modules would have to be installed on the instance manually, in order to allow the usage of its functionalities. The WebC-SaaS project proposes to improve this subject, by implementing an automatic deployment mechanism capable of
deploying and maintaining WebComfort Applications during a subscription period. To implement this, WebComfort lacked several mechanisms like Context Settings, Payment Methods and Instance Deployment which had to be developed. Next are presented the main challenges found preparing the WebC-SaaS infrastructure.

One of the main features of SaaS is the ability of its applications to support being accessed by various tenants on a single instance. Being this a central feature of SaaS, WebC-SaaS had to fulfill it. To accomplish this, SaaS applications implement multi-tenancy, which produce independent contexts of execution (virtual application instances) for each different client. However, this was a very complex system to be implemented from the beginning, and as such, a simpler adaptation of this technology was adopted in order to produce the same results. To accomplish this, contexts needed to be created on the WebC-SaaS platform. To model this, two concepts were introduced: the Sandbox and ContextSettings.

Besides multi-tenancy, another important feature of SaaS applications has to be its' configurability properties. As such, this is one of the main concerns of the project’s conception and implementation. Because WebComfort didn’t provide a uniform model of settings management to approach this issue, it was necessary to extend the Domain Model to capture new concepts capable of modeling context configurability on WebComfort Applications. To employ configurability in WebComfort it was identified that it was first necessary to create a mechanism to mark and classify what settings and values each WebComfort Module could define and assume, during execution. To model this, the VariationPoint concept was introduced. The other substantial challenge of implementing WebComfort configurability was to develop Settings’ storage and recovery for different contexts. WebComfort already implemented separate settings storage and recovery for Portals, Modules and Tabs, but these had a limited contribution in helping implement what WebC-SaaS had proposed. As seen in the topic before, implementing multi-tenancy in this project would require the possibility of retrieving different values for the same settings in different contexts.

5.1. Domain Model

Because WebComfort focused mainly on supporting Web Applications’ execution, it lacked the concepts necessary to define a customer service subscription and deployment context. For this reason, it was necessary to develop a Domain Model capable of capturing these concepts. In this section, the WebC-SaaS concepts and Domain Model are analyzed in more detail. To better understand the Domain Model of this project, concepts were divided into three packages, which are next presented:

Template package – To be able to deploy a service, several supporting concepts (i.e. templates) must be created, to store all the information necessary to generate a new service. This package groups these;

Application package –includes all the concepts related with accessing and managing existing services;

WebComfort package – This package is used throughout the documentation of the Domain Model, to group the concepts offered by the WebComfort platform;

Template package:
To be able to promote and deploy a service type, several supporting concepts must be created, to store all the information necessary to produce a new service subscription. In this package are represented these concepts.

Fig. 4. Template Concepts Overview.

As it can be seen, the ServiceLevelTemplate – which can be considered as the most important concept of this Package – is in the center of the image. The ServiceLevelTemplate represents the actual definition...
of the Service a customer subscribes to. Nonetheless, various ServiceLevelTemplates can be grouped to a ServiceTemplate, which represents a type of Service. What’s more, ServiceLevelTemplates must define one or more SubscriptionTemplates, which enclose the information necessary to produce a valid Subscription when a ServiceLevelTemplate is subscribed to. Finally, note that a SandboxTemplate is also present on this model view and has a one-to-many relationship with ServiceLevelTemplates. This is because ServiceLevelTemplates can produce new Services and deploy them to an existing Sandbox, so this makes the use of a SandboxTemplate facultative for some Services.

Note that the SubscriptionTemplate, and SandboxTemplate concepts appear in this model in *Italics*, which means that these concepts are defined as abstract classes. This means that these concepts are only generic types and are mainly used to define an interface that must be implemented by other classes. This approach allows employing a Strategy design pattern to implement different behaviors using a shared interface. This on the other hand, allows concepts that contain these generic object types to invoke specific processes – declared in the generic abstract class – without knowing the concrete behavior that that object type is going to assume.

**Application package:**

After deploying applications on the WebC-SaaS platform as services, it is necessary to maintain these concepts to be able to manage and validate their behavior on the platform. This package groups these concepts.

![Application package](image)

*Fig. 5. Application Concepts Overview.*

As it can be seen in the picture above, the main concepts captured in this package are the Service, Subscription, Sandbox a Customer classes. These concepts assemble all the information relative to a service subscription, all its functionality and configuration, the validity of the subscription, and who can access what and where in the platform.

This view reveals the relations between the main concepts of this Package. As it can be seen, every concept in this view relates to Service and Customer objects. What’s more, there are generally two types of associations between Customers and the other concepts. This is due to the fact that Customers can behave as Administrator and User abstractions.

Note that the Subscription, Sandbox, and Customer concepts appear in this model in *Italics*, which means that these concepts are defined as abstract classes. This means that these concepts are only generic types and are mainly used to define an interface that must be implemented by other classes. This approach allows employing a Strategy design pattern to implement different behaviors using a shared interface. This on the other hand, allows concepts that contain these generic object types to invoke specific processes – declared in the generic abstract class – without knowing the concrete behavior that that object type is going to assume.

**6. Implementation**

To implement the WebC-SaaS infrastructure, WebComfort components like Modules and Extenders were used to impart the new functionalities and concepts of the project into the platform. Following, an overview of the implemented WebC-SaaS components is presented and next, a more detailed description of these components is made.
SaaS (Software As A Service) – Models and Infra-Structures
Dissertation’s Course Extended Summary 08/09

The image above represents the structure of the several types of WebComfort and WebC-SaaS components implemented throughout this project. Note that all components depend on the Utilities object. This object represents a static class that served as a micro-kernel for many simple but useful operations realized throughout the project. Note also that this project has also implemented CustomControls, which are used throughout the development of Pages and Modules. The API is used by Pages, Modules and Providers, and apart from that, Providers depend only of the Pages Package. Next, a more detailed description of the Modules and Pages components is made.

Modules

During the development of this project, it was necessary to develop various WebComfort Modules to impart the WebC-SaaS functionalities to the WebComfort platform. In the image below these modules are enumerated and organized in packages.

- **Customers**
  - CustomerInfo
  - CustomersInfo

- **Subscriptions**
  - SubscribedServices
  - SubscribedServicesAdmin

- **Templates**
  - TemplatesManager

In the Customers package are included the CustomerInfo and CustomersInfo modules. These modules both return information about Customer objects, but the latter is employed in an administration context, and can be used to manage all customers on a WebC-SaaS platform. These modules can also be used to configure the customers information, as well its Organizations.

In the Subscriptions package are included the SubscribedServices and SubscribedServicesAdmin modules. These modules both return information about Subscription objects, but like in the Customers package, the latter is employed in an administration context, and can be used to manage all the subscriptions on a WebC-SaaS platform. These modules can also be used to configure a subscription, as well as disable and enable it. Note also that this package also includes the Subscriber module which was implemented to promote the available service types in a platform.

In the Tempaltes package is included the TemplatesManager module, used to add, delete and edit ServiceTemplate definitions.

Finally, to manage a customers’ sandbox, a user can access the SandboxRoles and SandboxTabs to manage the Roles and Tabs of a single Sandbox context.

Pages

To create and manage many of the concepts of this project, WebComfort Modules were not adequate, so WebComfort Pages were also employed to define workflows. In this section, two of the most important workflows of this project are described in detail – the Templates Management Workflow and the Subscription Workflow. Other pages are also presented to enumerate all the pages.
involved in this project. In the image below all of these pages are enumerated and organized in packages.

Fig. 8. WebC-SaaS pages.

**Extenders**

To support many of the functions of the WebC-SaaS platform, another structural component needed to be added to the project – WebComfort Providers. These components represent the boundary of a platform. As such, these concepts were used to manipulate the normal workflow of a WebComfort instance. In the image below are enumerated the implemented Providers components.

Fig. 9. WebC-SaaS Providers.

Of these components, the SaaSPortalExtender is the most relevant to this project. In this component is defined a class that extends the WebComfort Extender class. This makes the SaaSPortalExtender class a device for capturing WebComfort custom platform events, as well as the Begin and End events of a standard Web Application. This device allowed implementing subscription validation and revoking; detection and treatment of WebComfort events like Module or User creations.

Furthermore, this abstraction supported altering some additional aspects of the Portal, including: adding subscription links to the banner of the site; changing the behavior of the Home link button in order to redirect to the users’ main sandbox Homepage; change the page’s title and the banner Tabs, when in the context of a particular sandbox.

7. **Validation**

To validate the WebC-SaaS project and its new features, several Service types were designed and deployed in a production environment with the intention of promoting and collecting information about these features.

In order to gather important feedback from these tests, it was intended to evaluate this input by conducting a workshop or an inquiry with the users that tested the platform. Users would be asked to complete several tasks over the application – namely the subscription and access to a service –, and in the end to fill up an inquiry to assess the level of functionality and satisfaction with the platform. However, due to the fact that the WebC-SaaS project works in great part transparently to the user – as opposed to the actual applications that WebComfort support –, these inquiries would be hard to be unambiguous to the user. Users would likely tend to evaluate the actual WebComfort platform and applications instead of WebC-SaaS. Due to this, this method of evaluation was discarded. Nonetheless, because many WebComfort application’s features were related to management of web contents, a validation scenario was to create a service capable of assembling those application modules, so as to provide users with a way to try these on a trial basis.

Another possible test would be to evaluate the acceptance and functionality of the implemented solution from an authentic customer’s perspective. Moreover, the customer in question would have to already be familiarized with the WebComfort platform and its functionalities. This way, the customer would not likely evaluate the actual WebComfort platform which he already knew, but the new features introduced by the WebC-SaaS project, including service subscription, sandboxing and context settings. Searching for a customer of this type, a promising test case was found which consisted in integrating WebC-SaaS on the WebTrails project. After this integration, the validation would consist in soliciting the
customer to perform a group of tasks related with the service management, and finally evaluate the easiness, functionality and acceptance perceived by the user.

8. Conclusions

Creating a software product or web-based application is relatively easy. Making it a viable, profitable, and sustainable business is more challenging. Every software business model has its inefficiencies: Legacy models are hardly scalable and costly, but are familiar to the user and give more control to the client; ASPs are cheaper and more scalable but take control from the client. On the other hand, the object of study of this thesis – the SaaS model –, presents itself as an ASP successor emending its inefficiencies and exploiting the Internet as a platform, but it still takes more control from the client.

In effect, the Internet creates new ways to promote, distribute and support software. Due to the dynamicity of this channel, applications can be developed to deliver services massively to innumerous users in any part of the world, while maintaining high levels of availability. Given the constant evolution of the Internet nowadays, approximating software development to this channel is becoming more and more common and viable for various businesses. Among other advantages, the Internet as platform allows customers to easily experiment the applications on a trial instance, easily integrate those services with the customer’s system, access the applications over a common browser and easily maintain the application. On the other hand, this channel has a big fault, which is its dependence of a network connection to the Internet. Because of this, it is not always best to employ SaaS solutions to solve one’s problem; as pondering is needed for each case.

Given the uncertainty of today’s global marketplace, Market Intelligence specialists like IDC confirm the tendency of businesses to cut on IT expenses to focus on their core business and conclude that until 2012, IT Outsourcing will have the biggest development in the IT business segment. Moreover, these analysts foresee that due to the deceleration of IT investment, software business companies will have to focus on exploring new ways to deliver their applications.

In the last few years, the SaaS model has appeared as a qualified model to respond to these challenges, and has since become a reference model in the school of thought on how software should be effectively delivered.

During this work, it’s been proved that WebComfort is a robust and very extensible framework. Due to this, implementing the proposed WebC-SaaS infrastructure was made possible on this platform. As a result, this project has accomplished in putting into practice most of the SaaS features that were proposed to be implemented: Services definitions and Subscriptions are now possible over the WebComfort platform; and what’s more, existing applications don’t necessarily have to be altered to harness these features.

As this work’s conclusion, it is probable that given the current technological advances and financial crisis, traditional software vendors will have to adapt to the new technological and business reality. The SaaS and ASP family of models propose alternative approaches to solve many traditional issues. For that reason, these models are feasible choices for both software vendors and customers to overcome their problems. On an optimistic note, this work’s author suggests the SaaS model has an especially favorable outlook for the near future.
Acknowledgements

Many thanks to my professors at Universidade Tecnológica de Lisboa – Instituto Superior Técnico: Professor Alberto Silva, my thesis supervisor for offering valuable advice; to João Saraiva, my accompanist supervisor for introducing me to the WebComfort platform and giving guidance on how to approach technical issues.

Many thanks also to my university colleagues and friends for sharing their thoughts and helping me keep motivated.

And finally, never enough thanks to my family that has guided me on my education formation.

References

10. WebComfort (http://www.webcomfort.org/Portal)
12. SIQuant (http://www.siquant.pt/)