



INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa



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

João Samuel Nunes Lopes

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática e de Computadores

Júri

Presidente: Prof. José Manuel Nunes Salvador Tribolet, Departamento de
Engenharia Informática (DEI)

Orientador: Prof. Alberto Manuel Rodrigues da Silva, Departamento de Engenharia
Informática (DEI)

Vogais: Ademar Aguiar, Faculdade de Engenharia da Universidade do Porto (FEUP)

Outubro de 2009

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.

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

Resumo

SaaS (*Software as a Service*), frequente mas incorrectamente referido como o modelo ASP (*Application Service Provider*), é actualmente considerado por muitos como a revolução na área de distribuição de software. Tendo em conta a evolução dos novos canais de comunicação como a Internet hoje em dia, muitos acreditam que aplicações tradicionais, *of the shelf*, brevemente tornar-se-ão obsoletas em comparação a aplicações *web*, outsourcing de produtos e serviços que removem a responsabilidade de instalação, manutenção ou suporte. Enquanto estas previsões drásticas ainda não se vieram a confirmar, a essência desta mudança – a distribuição, manutenção e pagamento do software como um serviço em vez de como um produto – está a afectar todos os intervenientes na indústria do software. Este projecto estuda e compara os modelos tradicionais, ASP e SaaS; assim como implementa a infra-estrutura SaaS sobre a plataforma WebComfort.

Palavras-chave

Negócio; Serviços; Outsourcing de TI; Modelos de Distribuição de Software; Internet; Valor Acrescentado

Table of Contents

- Acknowledgements 1
- Abstract..... 2
- Keywords 2
- Resumo 2
- Palavras-chave 2
- Table of Contents 3
- List of Tables and Images 5
- Abbreviations and Glossary..... 5
- 1. Introduction..... 6
 - 1.1. Overview 6
 - 1.2. Context 8
 - WebComfort..... 8
 - 1.3. Problems 9
 - Business Issues..... 10
 - Technical Issues 10
 - 1.4. Objectives..... 11
 - 1.5. Document Structure 12
- 2. State Of The Art..... 13
 - 2.1. Legacy models 13
 - Operational characteristics 14
 - Integration..... 14
 - Maintenance, Support and Updates 15
 - Application access..... 15
 - Licensing..... 15
 - Payment model..... 16
 - Intellectual property protection 16
 - Service Level Agreements (SLAs) 16
 - Market targets..... 16
 - Marketing and product promotion..... 17
 - Unresolved issues 17
 - 2.2. ASP model 18
 - Operational characteristics 19
 - Integration..... 19
 - Maintenance, Support and Updates 20
 - Application access..... 21
 - Licensing..... 21
 - Payment model..... 21
 - Intellectual property protection 22
 - Service Level Agreements (SLAs) 22
 - Market targets..... 23
 - Marketing and product promotion..... 23
 - Unresolved issues 24
 - 2.3. SaaS model..... 25
 - Operational characteristics 25
 - Integration..... 26
 - Maintenance, Support and Updates 27
 - Application access..... 28
 - Licensing..... 28
 - Payment model..... 28
 - Intellectual property protection 29
 - Service Level Agreements (SLAs) 30
 - Market targets..... 30
 - Marketing and product promotion..... 31
 - Unresolved issues 31

2.4. Discussion	32
<i>Legacy</i>	32
ASP	33
SaaS applications	34
3. WebC-SaaS – Conception	35
3.1. Actors and Use Cases.....	35
Actors.....	35
Use Cases	37
3.2. Requirements	38
3.3. Challenges	40
Multi-tenancy	40
Context Settings and Variation Points.....	41
Integration with WebComfort.....	41
Payment Methods.....	42
Instance Deployments and Maintenance	42
3.4. Concepts and Domain Model.....	43
Domain Packages	43
Template Package.....	44
Application Package	49
WebComfort Package	55
4. WebC-SaaS – Implementation.....	56
4.1. Integration	57
4.2. Modules	58
4.3. Pages	59
Templates Management.....	60
Subscription process	61
Sandboxes administration	63
4.4. Providers	63
4.5. Other Technical aspects	64
Context Settings and Variation Points.....	64
Payment Methods.....	65
5. Validation.....	66
5.1. Pages Services	67
5.2. WebTrails Services	68
6. Conclusion.....	69
6.1. Future Work.....	70
Instance deployment	70
Subscription to already deployed services	70
Integration with the WebComfort kernel	71
References	72

List of Tables and Images

Fig. 1. Extensibility of the WebComfort platform.	
Fig. 2. Relationship between WebC Users, Roles, Categories, Toolkits and ModuleDefinitions.....	
Fig. 3. Software distribution.....	
Fig. 4. WebC-SaaS Actors.	
Fig. 5. Service Administration use cases.	
Fig. 6. Package dependency.....	
Fig. 7. Template Concepts Overview.	
Fig. 8. Sandbox Templates View.....	
Fig. 9. Service Templates View.....	
Fig. 10. Subscription Templates View.	
Fig. 11. Applicational Concepts Overview.	
Fig. 12. Customers View.	
Fig. 13. Sandboxes View.	
Fig. 14. Services View.....	
Fig. 15. Subscriptions View.....	
Fig. 16. Context Settings View.....	
Fig. 17. WebC-SaaS components.....	
Fig. 18. WebC-SaaS integration with WebComfort.	
Fig. 19. WebC-SaaS modules.....	
Fig. 20. WebC-SaaS Module examples.	
Fig. 21. WebC-SaaS pages.....	
Fig. 22. WebC-SaaS templates definition.	
Fig. 23. WebC-SaaS subscription workflow.	
Fig. 24. WebC-SaaS Providers.	
Fig. 25. Homepage of a deployed Personal WebSite service.	
Fig. 26. WebTrails application, integrated with WebC-SaaS Services.	

Abbreviations and Glossary

ICT -----	Information and Communication Technologies
IT -----	Information Technologies
Legacy --	Traditional Software Business Models: Business models based on product licensing
OTS -----	Off the Shelf
SM -----	Software Manufacturers: Companies specialized in making software
ISV -----	Independent Software Vendor: Companies specialized in selling software
CMS -----	Content Management System
ASP -----	Application Software Provider Business Model
ASPs -----	ASP providers
SaaS -----	Software as a Service
PaaS -----	Platform as a Service
SLA -----	Service Level Agreement
VPN -----	Virtual Private Network

1. Introduction

1.1. Overview

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 [13].

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 outsource-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) [13].

Following the decline of the ASP model, the Software as a Service (SaaS) business model appeared as an ASP successor, to emend 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 [13]. 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.

1.2. Context

This thesis subject was initially proposed by Professor Alberto Manuel Rorigues da Silva, with the aim of studying the advantages and disadvantages of the various software distribution models, and in particular the SaaS distribution model. Furthermore, this proposal comprehended the possibility of improving the WebComfort platform by implementing the SaaS model over that framework. Following this proposal, an internship in the company SIQuant was suggested, which allowed to better focus in the study and implementation of this work.

WebComfort

The WebComfort platform is a Web Portal and Content Management System (CMS) Framework promoted by SIQuant [12]. It delivers contents to its users through any device with a common browser (e.g. IE or Firefox) over the Internet. The platform is developed using Microsoft's ASP.NET 2.0 technology (C#). WebComfort claims the separation between the content and its presentation. Content is presented through information modules, whose presentation can be configured without changing the data model and the underlining contents. In particular, regarding content presentation, a layout management mechanism is provided, at portal, tab and module levels [11]. Furthermore, WebComfort supports authorization and security policy management through a flexible role-based mechanism.

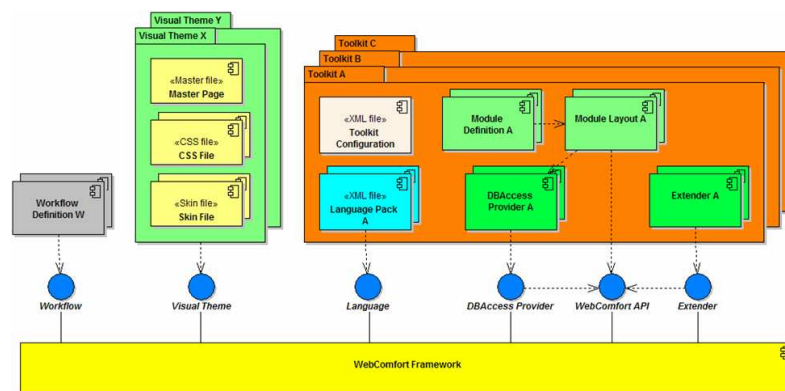


Fig. 1. Extensibility of the WebComfort platform.

The “Framework” designation comes from WebComfort’s easiness of extension, by allowing adding new module types to manage and display existent information, or even new types of information, supporting the development of new module logic and design through a well-defined module API. Modules are one of the elementary and most important components of the platform. They are responsible for embedding content and conferring functionality in the WebComfort application. They can communicate and share functionality between each other and can be integrated in WebComfort

Categories and Toolkits for easier deployment. WebComfort Categories are groups of module types defined at a portal level, which allow roles of users to add/edit/remove modules – and therefore content – to the portal. On the other hand, WebComfort Toolkits group module types at an applicational level, in order to allow installation of different module types in WebComfort instances. Other WebComfort platform features include [10]: visual themes support; multilanguage support; application extension support; workflow management support; data repository access; integration with Microsoft WebParts, among other things.

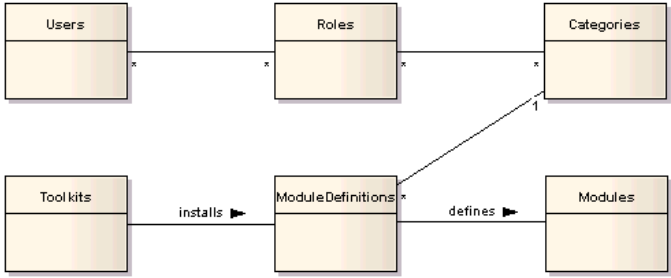


Fig. 2. Relationship between WebC Users, Roles, Categories, Toolkits and ModuleDefinitions.

WebComfort is more than a simple CMS. It is a matured and technically proficient framework, capable of deploying and managing Web Applications. What’s more, in spite of having several applications already implemented over this platform, a service subscription model is lacking. In this context, it is reasonable to choose WebComfort as a platform to develop the SaaS infrastructure and assess how customers perceive it.

1.3. Problems

In today's exceedingly competitive, 24x7 business world, IT must deliver applications to customers 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.

1.4. 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. This analysis will focus on, and consider the importance of particular parameters which will reflect on the following aspects:

- Operational characteristics – Identifies what the model offers and how it works;
- Integration – Considers the infra-structure necessary to run the applications/services and how they are integrated with the client's system;
- Maintenance, Support and Updates – How the applications/services are aligned to the client's needs and demands, and how they are kept up-to-date;
- Application access – Considers how the applications and services are accessed;
- Licensing – Deliberates on the general client's contractual duties and allowances;
- Payment model – Evaluates the fashion in which the payments are executed;
- Intellectual property protection – Considers the software vendor's intellectual property protection in each model;
- Service Level Agreements (SLAs) – Analyzes the type of SLAs issued;
- Market targets – Which markets the model can be applied to and its viability;
- Marketing and product promotion – Considers the marketing and technological mechanisms used by the providers to capture new and keep the existing clients;
- Unresolved issues – Identifies the main unresolved issues of each model;
- Additional considerations – Other important model aspects not considered in the latter topics;

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.

1.5. Document Structure

This document is divided in six sections:

Section 1. Introduction: This section is intended to contextualize the area of study of this project and clarify the problems and objectives of this work;

Section 2. State Of The Art: This section analyzes the State Of The Art of the *Legacy*, ASP and SaaS software distribution business models, and compares them discussing to evaluate their pros and cons;

Section 3. WebC-SaaS – Conception: This section describes the conception of the WebC-SaaS project. In this section, a description of the Domain Model is made in detail;

Section 4. WebC-SaaS – Implementation: This section presents the actual implementation of the WebC-SaaS platform;

Section 5. Validation: This section describes how the WebC-SaaS project was validated, and the results of this process;

Section 6. Conclusion: In this section, a conclusion of this work is made, and future work is suggested;

2. 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.

This project’s work reflects on the state of the art in software business, licensing and distribution models. To better understand these, this project will focus on three main groups of software distribution models – the traditional, ASP and SaaS models. 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.

2.1. 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.

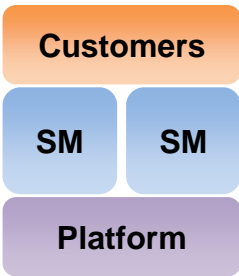


Fig. 3.1. Software distribution model (Legacy applications).

Operational characteristics

In this class of business models, software is developed as a product rather than a service. While this can be extremely profitable to software manufacturers and vendors, this isn't necessarily the case for its customers.

Traditional SM specialize on selling product licenses over services. Moreover, these licensing models can forbid customers to use an application solution copy on more than one device or user per license. For this reason, the larger the enterprises are that want to operate on these terms, the greater the expenditure on these products is. Nonetheless, different licensing plans can be discussed by the participants to reduce the cost of purchase and maintenance of these products.

On the other hand, from a software vendor's viewpoint, these models offer great financial advantages. While the development cost of a product may be high, the variable cost of sale is substantially lower, most times even negligible, when compared to other commodities. Furthermore, since software is the result of intellectual output of a programming team, it is difficult to accurately quantify its value. As a result, in a product oriented distribution model, the method to assess software value has been to align the price to the features that buyers generate value from. Consequently, these products don't produce a steady income revenue stream, but instead generate large amounts of capital from first time clients. To overcome this supply-side issue nonetheless, software vendors have evolved to provide additional services around their products, like consulting, maintenance, support, software solution customizations, or even develop new upgrades in exchange for supplementary fees. In reality, what can be observed is that major software enterprises, like the epitome of software product development – Microsoft –, has a strong set of services around its products.

Integration

In *Legacy* models, software is designed and developed to be installed and run on the customer's end-point devices. Each user works directly with a singular application copy in each equipment. Customers' IT teams have to install and maintain the software solutions in every device where they are needed, which can yield complex operations of IT management for larger enterprises. Due to this scalability issue, for this group of models and especially for larger enterprises, the existence of IT management teams is crucial.

Besides these devices, where the software is to be installed, the organization doesn't need additional infra-structure to run the applications.

Applications are deployed with the help of external data drives, like floppy disks, CD-ROMs or even Internet delivered executables, and may exchange data through channels of the same sort.

Maintenance, Support and Updates

Maintaining traditional application systems can be a tough and complex enterprise in any organization. IT teams must sustain program functionality in every device with an application installation. What's more, IT management requires the labor of specialized staff. For these reasons, and particularly if the business has little experience on the subject, IT management can be rather expensive. As such, for this kind of service an organization can opt to outsource IT management, whether purchasing this service from the software vendor or another provider.

Consistent revenues are available to software product firms from maintenance fees. The terms of maintenance fees differ, but they often provide customers with fixes, new features and support.

Maintenance fees are largely profit once the product is stable. A feature developed for one client – and paid for by that client – can be made available to all customers who pay the maintenance fee for that product.

Customization can be delivered to the customer over the form of patches and upgrades, which allows the program to be adjusted and aligned to the constantly changing customer's business requirements.

Application access

Traditionally-developed application solutions are self contained and therefore independent from third-party platforms. Programs are developed to be executed alone and are accessed directly by the end user.

Licensing

Software vendors sell products by selling their licensing agreements. Traditionally, vendors license products on a "one application per user" or "one application per device" approach, meaning that each user or device must work with a unique copy of the software respectively. As such, customers are obligated to purchase licenses for each user or device to whom or where the program is to be executed. Moreover, the same rules for a single application installment are applied to every copy of that software solution in an organization. As such, for a largely used product in an organization, this can become an expensive and ineffective requirement for customers. To balance this issue, alternative licensing and payment models based on the selling of multiple licenses (volume licenses) exist to grow customer profitability.

Another licensing model is based on hardware vending agreements. Since hardware sales can provide a good means for propagating software products, additional licensing agreements exist between hardware and software vendors to add value to their products.

Besides the allowance to use the core software product, *Legacy* licensing can also comprise additional services like software customization, maintenance, upgrades, support and consulting.

Payment model

While several different pricing schemes for software licensing exist, customers traditionally pay a flat fee for permanent product usage.

However, when considering additional product services like maintenance and support, payments are executed regularly based on contractual service agreements.

Intellectual property protection

Regarding intellectual property protection, this class of models has several software models with different approaches to this matter. The customer can decide between purchasing a copyrighted software solution to acquiring an open source solution.

Either way, the product of the intellectual work of software developers is usually protected by international law, and therefore the software products are also protected.

Nevertheless, in reality, *Legacy* models lack the mechanisms to monitor the users of these products. Consequently, some users exploit these products without paying for the software licenses.

Service Level Agreements (SLAs)

In this class of models, SLAs aren't discussed with the clients. Software is delivered as-is – as a product –, but supplementary services can be arranged around it. Nevertheless, these services have distinctive Service Level Agreements from the product and as a result are not relevant in this analysis.

Market targets

Software manufacturers that choose to develop software as a product rather than a service generally have to decide whether to build products for mass markets or niche markets. In the software business world, competition is extremely high and several successful businesses many times acquire

challenging new enterprises. As such, in a global market viewpoint, the safest strategy for a software vendor in this class of models is generally to target a niche market first.

In terms of market expansion, this class of models is somewhat limited. Software enterprises need to carry out great investments in order to try and execute market expansions.

When targeting a niche market, a software business should try to develop programs vastly oriented and aligned to its customer's business and business requirements. Later when expanding, software companies are best to operate on strategic vertical areas to their business, wherein they have more experience.

Furthermore, one of the main vital contributes for software businesses to thrive is the quality of service (QoS). A software vendor must be able to meet its client's needs and try to provide additional software improvements to encourage the customer to keep buying its products.

Marketing and product promotion

In using *Legacy* products, clients can be granted a trial period to decide on the purchase or non-purchase of the product. In this class of models, the client needs to install the software or a demonstration of the software to evaluate the application.

Products are promoted by various channels like word-of-mouth, the Internet, workgroups, consulting groups, among others; but one aspect is crucial in market acceptance, which is the quality of the product. Software solutions that provide what the client "needs" and that meet additional criteria of what the clients "wants" is a successful product, and many times constitutes an important selling mechanism.

Software is generally delivered in "box-wrapped" products and sold throughout various retail channels. As such, it is pertinent to develop the image of the product to its customers, as it is to offer good business opportunities to intermediate retail agents. Another channel through which products are sold is through agreements with hardware vendors. As such, it is equally important to find and maintain business opportunities with these agents.

Unresolved issues

One of the main issues, if not the biggest issue with this type of models is the fact that it is hard to apply in large scale environments, as well as it is costly to maintain. Another related problem is the fact that it takes much time to be integrated and upgraded. Software products and upgrades need to

be installed in every device where it is needed, and this takes a lot of time and effort. For a business with important Time To Market (TTM) requirements, this model fails on delivering the product quickly.

What's more, what can be observed in various cases is that the process of maintaining these systems continuously is insupportable, so businesses tend to reduce IT costs to focus on their core competencies. While this can be effective to a level if a business's IT system isn't habitually updated, in enterprises with higher IT requirements this strategy isn't viable, leading to inaccessible applications, loss of productivity and unhappy customers.

Another serious problem with this type of models concerns software licensing traceability. In reality, *Legacy* products and their licenses cannot be monitored continuously; so many software solutions are used improperly, disrespecting license agreements. For many years now, several software products have been usurped and used illegally by innumerable enterprises/people without software vendors being able to persecute them rigorously.

2.2. 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.

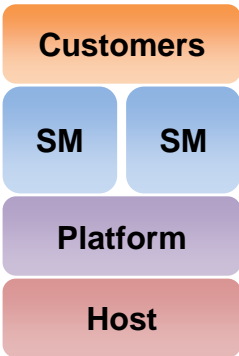


Fig. 4.2. Software distribution model (ASP).

Operational characteristics

The basic vision of the Application Service Provider model (ASP) is centered on separating software possession and ownership from its usage. As a result, ASPs provide software solutions as a service rather than a product. The core value of the ASP service is providing hosting, access and management of an application that is commercially available.

The main technological difference between this and other *Legacy* models is the use of application virtualization allowed by the introduction of computer networks. These, separate applications from hardware and software environments, as well as its users. In a virtualized environment, the core software product needs only be installed on a central mainframe – the server –, that is then accessed remotely by each user. Consequently, this leaves the greater part of the IT management process to be executed in the datacenter. This, in turn, allows for great savings in comparison to other *Legacy* models. Moreover, since the application is only in effect installed on a single central machine, providers have only to acquire a single license agreement for that device. The ASP model harnesses these advantages and removes the responsibility of IT management from the client.

Additionally, seeing as ASP providers specialize in applications hosting, customers can solicit their providers to run and deliver new applications of other commercial ASP products. This approach is particularly attractive for larger customers, since it poses a greater investment for the client. Moreover, in most cases the client has to own the software license to solicit these services.

The typical ASP provider offers several services with different Service Level Agreements (SLAs) in exchange for a monthly/annually fee. This approach allows the customer to have minimal service assurances and at the same time to better manage its IT budgets.

Finally, from a client's perspective, the ASP model has the potential to induce changes in terms of business processes and thus generally transforms the way customers do business.

Integration

In this model, applications are virtually separated from the hardware and software environments where they are executed. The core software solution is only installed on a central mainframe – the server –, that is then accessed remotely by each user through a client application over a network.

For ASP application integration, customers have to include an access to a network system. This system can range from a private corporate network to a simple Internet connection, but it has to guaranty the links between the central mainframe – where the application is being run – and the devices where the application is to be accessed. Furthermore, when integrating an ASP solution on a

private network, ASPs can provide the IT equipment necessary to run these systems, namely the datacenter where the applications are to be executed.

Because ASPs deliver easy scalable software solutions, installing applications is rather simple and immediate. Depending on the software product and service's requirements, ASPs have only to guaranty the installation of client applications – which typically have a short setup time – to access the core application hosted centrally. These client applications can vary between thin-clients – small applications with no business logic, that are limited to accessing the server and retrieving information from it (e.g. traditional browsers) – and fat-clients – bigger and smarter applications, with some business logic that access and interact with the server. For integrating an ASP solution, IT teams traditionally have to install proper client applications and configure network settings to access the mainframe. Nevertheless, these configurations generally happen only during the installation, since further vital settings can usually be set through the mainframe core application.

In detriment of this model though, due to the different technological approach from other *Legacy* products, ASP solutions can sometimes be less compatible with other traditional products. Given this, besides being more limited, applications can become unfamiliar to its final end-users.

Maintenance, Support and Updates

In terms of maintenance, support and updates, clients pay a regular fee to the service provider, and in turn, the ASP offers its customers the benefits of liberating their organizations from these technological complexities. Moreover, since the core application isn't installed on the customer's local machines, regression or compatibility issues are uncommon.

On the other hand, from a provider's point of view, IT teams can focus their core management tasks to managing the datacenter, where the instances of each application are executed. As a result, when the centralized application is updated, the updated application is delivered on-demand to every user; and in the same way, when an application is to be removed, it only needs to be uninstalled from the datacenter. What's more, in contrast to in-house management, this model allows for the provider to build more application management experience.

However, this model does present problems with customer support. Because ASP providers traditionally solely host intermediary application solutions instead of developing them, IT personnel has to take great effort into knowing the applications installed on their datacenters. Consequently, due to the immense variety of these intricate third-party application solutions installed on most datacenters, ASP providers have trouble answering to customer inquiries and providing business application alignments.

Application access

In the ASP model, applications are delivered through computer based networks (i.e. through an Intranet or the Internet). The application looks and feels like it is installed and running locally, but because its core application isn't installed locally, it reduces conflicts with other local applications. What's more, this model allows its users to access the applications they need, when they need them, with little effort.

Additionally, from a software development business viewpoint, this model permits the production of application solutions that can access data securely and locally. Data can be accessed locally through client applications installed on the customer's endpoint devices, and since applications can be deployed over private Intranets, security problems can be surmounted.

Licensing

This model forces applications to be bought in a similar way to *Legacy* models, through the purchase of software licenses. However, in this model, licenses allow the application host to execute the application and virtually share it with multiple users. Alas, ASP providers who want to offer full-service solutions that include virtual distribution have to pay software vendors for volume licensing programs that allow for third-party license use rights.

ASP providers have to own the software applications needed to provide the services. However, this business model can repay and compensate the provider through regular fees received from its customers. Since a single application installation can be shared by multiple customers, a single license agreement can be paid for by regular fees received from multiple customers.

As a result of this licensing model, clients don't have to obtain the product license directly, but instead pay for a service that is offered by the provider. Moreover, on the condition that the contracted Service Level Agreement and Software License allow that, customers can opt to provide their hired services to their intermediary affiliates.

Payment model

Traditional ASP providers sell large, expensive applications to large enterprises, but can also provide a per-use payment model for smaller sporadic clients. There can be identified two main payment models:

A pay-as-you-go model – where customers are billed for the right of entry and then are charged for the number of accesses to the service, which is usually a good choice for smaller enterprises;

A regular fee charge model – where customers are billed on a monthly/annually fee basis, which is particularly appropriate for larger enterprises;

The service has a low-cost of entry and is more or less expensive depending on the Service Level Agreement (SLA) conditions, and if the application is more or less used by users.

Regardless of this, either approach represents a renewable income source for the service provider and is more financially manageable for the customer.

Intellectual property protection

In terms of intellectual property protection and in accordance to the *Legacy* models, the ASP model allows several software protection approaches. As such, the provider can decide between purchasing and servicing a copyrighted software solution to acquiring and servicing an open source solution.

Depending on the selected software protection model, the software product is protected by international law, and therefore the service provider has to comply with the licensing contract. Moreover, the ASP provider has to guarantee that its clients comply with the software licensing.

As opposed to other *Legacy* models, this distribution model allows for a more rigorous product licensing compliance monitoring. Since from a software developer's viewpoint there are much less software solutions being licensed and even lesser clients purchasing these software licenses, the software vendor can more easily monitor its clients.

What's more, to the benefit of ASP providers, since services are not self-contained like *Legacy* products, users have more difficulty in exploiting a network service. The central application generally authenticates its users to guarantee these are registered and their subscription is still active. Alas, in this model, when a harmful user succeeds on bypassing the security obstacles, it can reach many customers' accounts and access their services and data.

Service Level Agreements (SLAs)

SLAs are the formal definitions of the level of service provided by a contracted service. This mechanism allows providers to legally assure their clients that a minimum level of service is delivered.

In the ASP model, SLAs are discussed during the negotiation of a service contract with a provider, in order to transfer the responsibilities of application hosting from the organization to that provider. These contracts generally entail financial penalties and the right to cease service supply and

provision, when one of the parts disrespects the agreement. Additionally, SLAs generally include the definition of the payment model, the service, performance assurances (e.g. Uptime percentage, Network speed, etc.), problem management assurances (e.g. network failure, disaster recovery, etc.), customer duties and the termination of the agreement.

Market targets

Traditional ASP providers used to offer several services with different Service Level Agreements (SLAs). Nowadays, ASPs usually target specific markets, and can therefore be classified as one of the following forms of ASP businesses:

Local or Consumer ASPs – deliver applications to individuals (e.g. E-mail application);

Functional or Specialist ASPs – deliver a single application to organizations (e.g. Credit Card processing applications);

Vertical market ASPs – deliver a solution package to a specific organization type (e.g. Inventory Management solutions);

Enterprise ASPs – deliver a wide variety of applications to organizations (e.g. CRM, KM, ERM, Tax analyses, etc.);

What's more, what can be observed nowadays is that nearly any expensive software application, including large applications like SAP are also deployed as ASP services to allow these companies to reach smaller customers affordably.

Marketing and product promotion

A great advantage from the usage of application virtualization is the possibility to add new users in minutes – generally without the need to install any application on further IT equipments. Moreover, if a client is interested in a particular ASP service, it can experiment that service before subscribing it.

Products are promoted by various channels like word-of-mouth, the Internet, workgroups, consulting groups, among others; but one aspect is crucial in market acceptance, which is the reliability of the provider. The customer relies greatly on the provider to service its functional needs. As such, inadequate servicing can lead to business liabilities. It is then pertinent to develop the image of the provider, as to stimulate trust to customers.

Unresolved issues

From a client's perspective, the main issues with ASP service provision are linked to security issues. The customers delegate their IT systems to a third-party entity, so they become greatly dependent of their provider. Although a client can sign security assertive SLAs, when a client accepts to be supplied by a service provider it loses control of its data, and due to that, loses control of corporate image. If an ASP leaks, corrupts, or has insufficient security to protect client information, that customer in turn loses business credibility. To correct some of these security issues though, customers can solicit ASP providers to be audited for security certification. Moreover, clients can request that the application be run on the customer's house and/or be accessed via a private network, therefore reducing further security risks.

Additionally, when accepting ASP service provisioning, the client must generally accept the application as it is serviced by the provider. The provider can only afford customized solutions for larger clients. What's more, after ASP solutions are integrated, clients might have difficulties to adjust to the new and different application technologies.

Another problem is that clients might become too dependable of ASP providers, relying on the providers to supply critical business functions, thus limiting their control of those functions.

Giving support to a client can also become a hard task for an ASP provider. Given that ASP providers generally host third-party solutions, their IT personnel has to take great effort into knowing the applications installed on the datacenters. Consequently, on the condition that the provider has a vast variety of complex third-party application solutions installed on its datacenters, it has difficulty answering to customer problems and providing application customizations.

Additionally, another grave issue is the fact that these services are all dependent of network systems. In the eventuality of a network congestion or failure, the systems cannot access the central mainframe where the core application solution is executed. As a consequence, the client applications may not be able to operate normally.

Another issue related with ASPs is the fact that changes to IT businesses can affect the types of service delivered and their service levels.

2.3. SaaS model

Software as a Service (SaaS) is a subset model of Everything as a Service (EaaS). 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.

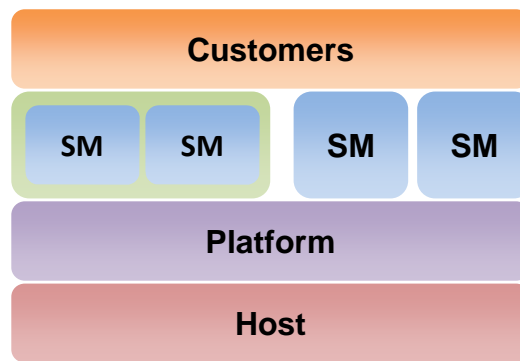


Fig. 5.3. Software distribution model (SaaS).

Operational characteristics

Like its predecessor – the ASP model –, in SaaS, software applications are delivered as services. The difference between the two is that ASP enterprises narrow their business to providing application hosting of commercial applications, and in SaaS, vendors constantly innovate to remain competitive. Furthermore, by being able to monitor system performance and user experience, SaaS is capable to anticipate alignment problems and quickly address them should they occur. Consequently, there is a shift of focus from the supply-side to the customer-side. The innovation may be in the form of better processes, better hiring strategies, or in developing value-added complex technical solutions for their clients.

To allow sharing applications with its customers, the SaaS model proposes a slightly different software architecture methodology to the ASP model approach. To be able to share an application and achieve better performance, SaaS applications implement multi-tenancy. Multi-tenancy refers to a principle in software architecture where a single instance of the software runs on a machine and serves multiple client users – the tenants. Multi-tenancy contrasts with the ASP multi-instance architecture, where separate software instances are set up for different client organizations. With a multitenant architecture, a software application is designed to virtually partition its data and configuration so that each user works with a customized virtual application, but sharing the instance. This methodology allows to achieve better economies of scale while practicing more competitive prices.

From a customer's perspective, SaaS applications are accessed through simple browsers and are presented to the as dynamic web pages. Nonetheless, services can be delivered as alternative forms (e.g. XML-based mash-ups, RSS or Atom feeds, email digests, etc.). Given the openness of this model, and following the recent wave of Web 2.0 applications, SaaS services can interact with many services throughout the Internet, and moreover can access other SaaS services likewise over this channel. As such, the provision of SaaS services leaves the possibility of future services to be composed of diverse SaaS services, allowing different services to be connected and supplied as a larger solution. Alas, this software model also has its disadvantages, as browsers entail limitations to some *Legacy* application features.

The typical SaaS provider offers a determined set of services with different Service Level Agreements (SLAs) in exchange for a monthly/annually fee. This approach allows the SaaS provider to reach a wider range of customers and better manage its services and customer's needs; and the customer to have minimal service assurances and at the same time to better manage its IT budgets.

Additionally, another advantage of the SaaS model is that applications are compatible with the ASP provisioning system. After the ASP model proved to be insufficient, many providers had to change their business strategy. As a result, several ASP providers saw in SaaS an opportunity to take advantage of the ASP synergies and provide both types of service. Likewise, SaaS software vendors saw an opportunity in ASP providers to promote sell and service their services.

Finally, from a client's perspective, the SaaS model has the potential to induce changes in terms of personal user's processes and thus generally transforms the way its users work.

Integration

Because SaaS providers deliver easy scalable software solutions, integrating applications is rather simple and immediate. While traditional software runs inside a firewall, SaaS applications run

entirely remotely and are available to all sorts of independent users and organizations. Unlike other traditional service providers – like the ASP model –, SaaS does not install any code outside the provider, and every aspect (e.g. content, service, support) of SaaS is delivered through the Internet.

Traditionally, users are not required to install any software. They only need to have a common web browser (e.g. IE, Firefox, etc.) installed, but most operating systems already provide these applications. As such, the user needs only to connect to the Internet and access the service.

Additionally, due to the radical technological approach to deliver applications through a browser, compatibility with other traditional products may be an issue. Since the browser is a relatively simple thin-client program, SaaS solutions can sometimes be inadequate to perform some solutions and are less compatible with other traditional products. Moreover, the delivered User Interface (UI) can be unfamiliar to its final end-users.

On the other hand, since much of a company's data resides inside its firewall, there is an integration challenge to connect local with remote systems. Differently from the ASP model, integrating a SaaS application with local systems is an entirely different job from integrating two local systems which can be both customized. Since browsers currently lack the tools to deliver an ideal solution for local data access, SaaS applications still present a significant weakness in this matter.

Maintenance, Support and Updates

Like in the ASP model, updating the software is a SaaS provider internal task. Clients pay a regular fee to the service provider in order to discard the technological complexities associated with application maintenance.

Like in ASP, from a SaaS provider's point of view, IT teams can focus their core management tasks to managing the datacenter, where the application is actually run. What's more, SaaS applications provide simple personal user interfaces, so configuring the application can generally be done by the customers. Furthermore, because SaaS applications are not installed locally, they don't affect other local applications. Applications are merely reset if a problem occurs as opposed to being uninstalled and re-installed. What's more, new applications can be streamed at any time, upon user requisition. As such, providers can focus on maintaining the IT equipment and providing much appreciated client support and business alignment. Seeing as SaaS applications are constantly being improved and upgraded, the collection of application advances are constantly being changed for the benefit of every client.

Application access

Similarly to the ASP model, SaaS applications are delivered through a network, but in this case the network is almost always the Internet and services are accessed through common browsers.

Accessing SaaS applications, users need only run a browser, load the provider's link, and enter a simple login. A user can configure it without ever physically installing the software. Nothing gets shipped or installed, nor are there issues of compatibility with existing systems. Moreover, users can access the applications they need, when they need them, with little effort.

Considering application access, the only weakness with this approach is accessing data securely and locally. As most common browsers do not generally follow web standards, they do not yet provide a uniform mechanism to access data. Since SaaS applications are all delivered through browsers, data cannot be accessed locally, and since applications and data are deployed over the Internet, security problems can arise from data interception (e.g. sniffers, bypassing datacenters, etc.).

Licensing

Licensing is actually easier in SaaS because access to the application can be controlled by the provider. As such, the license isn't sold like in *Legacy* models and resellers don't carry any title, but rather provide the application as service.

As in ASP, SaaS clients don't have to obtain the product license directly, but instead pay for a service that is delivered by the provider. Moreover, on the condition that the contracted Service Level Agreement and Software License allow that, customers can opt to provide their hired services to other intermediary affiliates.

Payment model

The service has a low-cost of entry and is more or less expensive depending on the Service Level Agreement (SLA) conditions, and sometimes if the application is more or less used by users. Next is a description of the various SaaS payment plans:

Fixed-Fee model – users pay a predetermined monthly fee based on the number of users supported, what application modules are rented and what service and support levels were specified by the customer;

Subscription-Based model – users pay a monthly payment calculated taking into account the actually used software, and includes a commitment as to the actual number of users. Subscriptions are usually written on a per-seat or named user basis;

Usage-Based model – users pay a monthly payment determined by application usage and is typically related to peak or near-peak levels of usage. Customers pay for the total processing requirements to run the application;

Transaction-Based model – users pay a monthly fee based on the application usage and is relayed business transactions (e.g. for online scheduling and similar services).

Value-Based, Shared Risk or Revenue model – users pay a monthly fee according to whatever software is needed to achieve business goals. The price for the service is linked to the achievement of the customer's goals;

Regardless of this, either approach represents a renewable income source for the service provider and is more financially manageable for the customer.

Intellectual property protection

In terms of intellectual property protection and in accordance to the other *Legacy* models, SaaS allows several software protection approaches. As such, the provider can decide between purchasing and servicing a copyrighted software solution to acquiring and servicing an open source solution.

Depending on the selected software protection model, the software product is protected by international law, and therefore the service provider has to comply with the licensing contract. Moreover, the provider has to guarantee that its clients comply with the software licensing.

Furthermore, the SaaS model offers a more extensive intellectual property protection. In this model, the software developers' interests are protected. Since SaaS services aren't sold directly, software developers are rewarded on a regular fee basis. In turn, developers are more rigorously paid for their actual services. What's more, as an outcome of this, there is a continuous investment on the development of the software, leading to more effective customer-oriented applications.

As in ASP, this distribution model allows for a more rigorous product licensing compliance monitoring. Since from a software developer's viewpoint customers don't purchase licenses, software vendors can more easily monitor its clients.

Differently from traditional products, network dependent services are generally harder to hack without previously subscribing to them, since they are not locally present. Conversely, even though SaaS providers register their authenticated users, harmful users can exploit the services over the Internet. Alas, when succeeded, these users can reach many customers' accounts and access their services and data.

Service Level Agreements (SLAs)

SLAs are the formal definitions of the level of service provided by a contracted service. This mechanism allows providers to legally assure their clients that a minimum level of service is delivered.

In the SaaS model, SLAs are generally preconfigured and wholesaled to various users. Like in the ASP model, these contracts can entail financial credits and the right to cease service provision, when one of the parts disrespects the agreement. Additionally, SLAs generally include the definition of the payment model, the service, performance assurances (e.g. Uptime percentage, Network speed, etc.), problem management assurances (e.g. network failure, disaster recovery, etc.), customer duties and the termination of the agreement.

Market targets

The SaaS model's primary market targets are standard consumers and small to medium enterprises (SMEs), but really anyone with an Internet connection is a target. What's more, frequently the buyer is the user rather than a business's IT department.

SaaS providers can be classified as one of the following forms of ASP businesses:

Personal providers – deliver applications to individuals (e.g. E-mail application);

Collaborative market providers – deliver a solution package to users of a specific organization type (e.g. Inventory Management solutions);

Enterprise providers – deliver users a wider variety of enterprise applications (e.g. CRM, KM, ERM, Tax analyses, etc.);

Since SaaS is a relatively new, competitive business, providers have to have a well defined set of services to succeed. Moreover, most SaaS providers don't venture in other areas of business. Given these facts, these SaaS providers can be considered as "Personal" providers.

What's more, what can be observed nowadays is that many major software companies, including the epitome of software development Microsoft, are also adopting their applications to new SaaS-based models of their own (e.g. Microsoft's S+S), to preserve market leadership.

Marketing and product promotion

Like the ASP model, SaaS benefits from marketing advantages the Internet provides. Services can be promoted easily and offered to anyone on a trial basis. What's more, since SaaS providers are open and recurrently conscious of what clients demand, they can promote service improvements quicker and more effectively to new clients than in other alternative models.

Given that SaaS services are low-priced and directed especially towards numerous standard consumers and SMEs, direct market selling can become expensive. As such, these services are particularly promoted by channels like word-of-mouth and the Internet, among others. Like in ASP, when promoting SaaS services, several aspects are important but one can be identified as being most important, which is provider reliability. The customer relies on the provider to service its functional needs. As such, inadequate servicing can lead to business liabilities. Furthermore, since SaaS services are especially intended to be provided to innumerable users over the Internet, it is also crucial to maintain good performance, scalability and availability levels so the delivered service doesn't become mediocre.

Unresolved issues

From a client's perspective, the main issues with SaaS – in the like to the ASP model – are linked to security issues. The customers delegate their IT systems to a third-party entity, so they become greatly dependent of their provider. Although a client can sign security assertive SLAs, when a client accepts to be supplied by a service provider it loses control of its data, and due to that, loses control of corporate image. In SaaS, service providers are given access to sensible private data which can be maintained in external datacenters. For this reason, the security of data and the reliability of the service provider are matters of great importance. If a SaaS provider leaks, corrupts, or has insufficient security to protect client information, that customer in turn loses business credibility. To prevail over this issue, it is essential to certify and audit the provider.

Multi-tenancy can also be hard to implement and maintain. It is a crucial feature of SaaS applications for these to work, but can endow user related security and configuration problems. These solutions can compromise a business's privacy if a harmful tenant succeeds in assuming another tenant's identity. Moreover, like in ASP, when a harmful user succeeds on bypassing the security

obstacles, it can reach many customers' accounts and access their services and data. Nonetheless, these security issues can be somewhat surmounted by extensive testing and security audits.

Another problem with this model, like in ASP, is the client's possible dependability of the service providers on critical functions. Since they rely on providers to supply these functions, these IT solutions limit their control of those functions.

Another problem with this model is the difficulty in accessing local data. Despite the fact that the ASP model allows providers to install client side applications, the SaaS model dictates that the application should only be delivered through common browsers. Seeming as these thin-client applications do not yet provide a standardized mechanism to access local data securely, the model is somewhat limited in this issue.

Furthermore, for the SaaS model to work, SaaS providers are required to define their set of services well, so that it can achieve economies of scale and balance supply and demand issues. Given that this requires areas of IT that are ubiquitous and commodity-like, single SaaS solutions are not suitable for innovative or highly specialized niche systems, though SaaS may be used to provide one or more services in such systems.

Another problem is the fact that, since SaaS solutions have to run over simple browsers, users might lose some functionality – only available in bigger traditional client applications –, so the interaction with other software products may not be so efficient.

Another grave issue is the fact that these services are all dependent of network systems. In the eventuality of a network congestion or failure, users cannot access efficiently or effectively the provider's services.

2.4. Discussion

In this section, the previously studied Business Models are compared, to discuss the various advantages and disadvantages of those models, and assess in what circumstances each of these models is generally more adequate.

Legacy

Pros

As previously seen, *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.

Cons

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.

Conclusion

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.

ASP

Pros

A great advantage of this 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).

One feature of ASP is its dynamicity 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.

Cons

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.

Conclusion

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

SaaS applications

Pros

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.

Cons

One of the cons of SaaS is the fact 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.

Conclusion

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.

3. WebC-SaaS – Conception

As described previously, this work has a practical objective of implementing the SaaS infrastructure over the WebComfort Web Application Framework. To identify it, the project was named as WebC-SaaS, respecting WebComfort Toolkit naming conventions. Next is presented the conception of the project, along with the requirements and challenges of implementing it over the WebComfort platform, taking into consideration the existing infrastructure and applications of this robust framework.

3.1. Actors and Use Cases

Before starting the conception of this project, it is necessary to comprehend how this project will be used and by who. As such, next are presented the identified actors and main use cases of the WebC-SaaS platform:

Actors

There is a hierarchy of key actors supported in the SaaS model system. One of the first tasks of this project's work consisted in identifying these actors and draw the actor hierarchy. Following is a summary of the identified actors and their hierarchy:

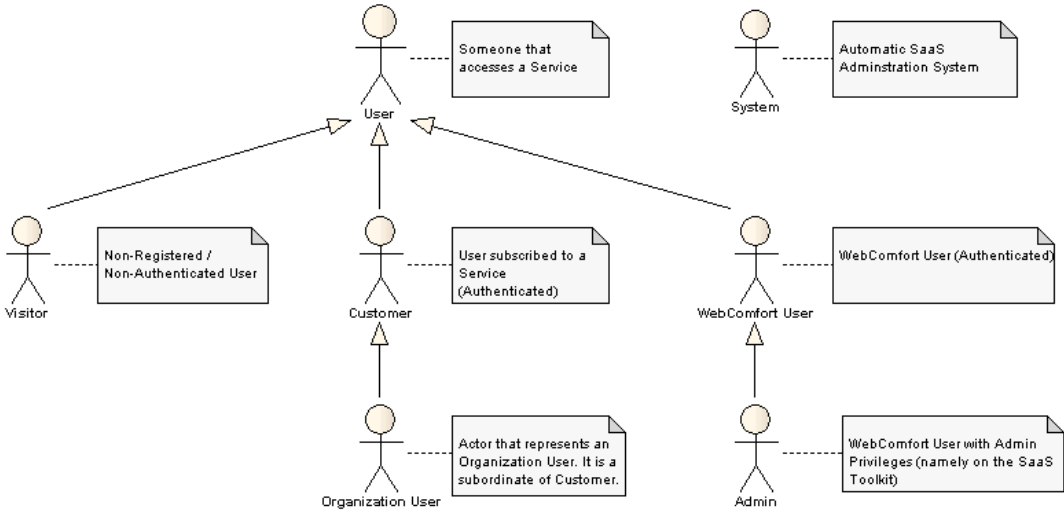


Fig. 6. WebC-SaaS Actors.

System – This actor represents the system and is responsible for maintaining the application environment coherent. It is responsible among other things to detect and unsubscribe users from expired services.

User – This actor is the generic human actor. It represents any registered or non-registered users, WebComfort users or SaaS users.

Visitor – The Visitor actor represents a person which might be a WebComfort user or a SaaS user, but is not yet authenticated in the system. This user is only allowed to register/authenticate and to perform other basic interactions with the system.

Customer – This actor represents the service subscriber. It is the service administrator for a particular application instance. It can access and configure the system's services, as well as control possible subordinate users and their privileges.

Organization User – The Organization User actor represents the subordinate users of an organization. These users can only access a subgroup of the services the Organization has subscribed. Note that Organization Admins also exist to administer Organization service subscriptions, but in this context are considered to function as a Customer actor.

WebComfort User – This actor corresponds to a generic WebComfort user. It cannot access SaaS services until it registers as a Customer. After registering, an authenticated WebComfort User is considered as a Customer.

Admin – This actor represents a WebComfort administrator. Besides being able to administer the WebComfort platform and its intricacies, the admin is responsible for creating Template Services and defining its details and Levels of Service (e.g. SLAs, pricing, description, etc.).

Use Cases

Various Use Cases were identified in this work. In order to better comprehend the use cases, these were grouped by similar functionality. Below, are summarized some of the most relevant use cases, related to Service Administration.

Consult Services – The Admin can consult services. This allows consulting information about any service subscription. Furthermore, the user can change the information related to a service's instance and access and manage its set of subscriptions and subscribers;

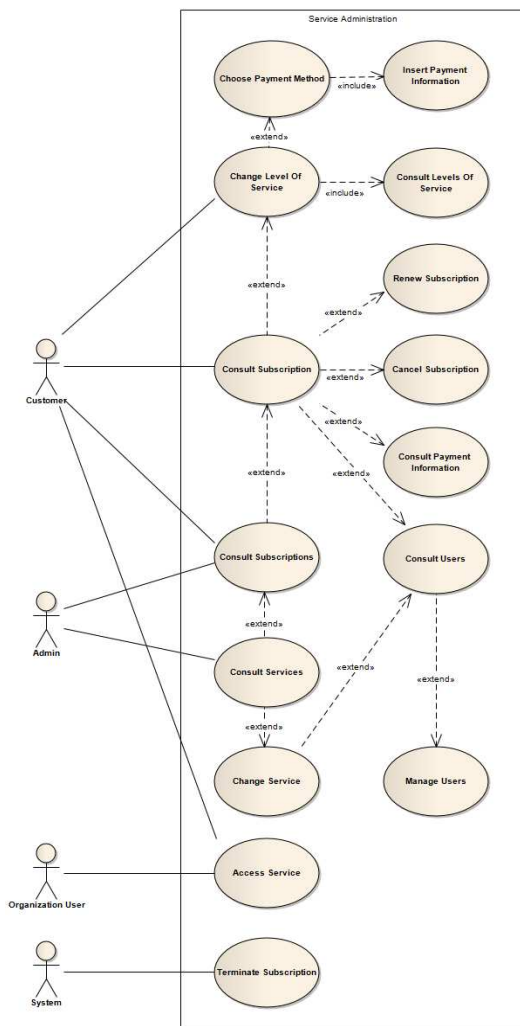


Fig. 7. Service Administration use cases.

These and more use cases can be viewed in more detail in the HtmlReport folder included on the CD annexed to this project presentation.

Change Level Of Service – Customers can alter a service's Level Of Service on the run. For this, it may be forced to pay for these changes.

Consult Subscription – A Customer can access a running service's subscription and renew or cancel it, change its Level Of Service, consult other subscription information or manage the set of users with service permissions.

Consult Subscriptions – A Customer can see and access all its subscriptions and control various aspects of each subscription (see Consult Subscription).

Access Service – Any subscriber (Customer or Organization User) can access a service, on the condition that it has permission to access that service.

Terminate Subscription – The System has to guarantee that no Customer can access an expired service. For that it has to terminate subscriptions when the conditions for that are met.

3.2. Requirements

Before being able to realize the WebC-SaaS conception, it is first necessary to understand the goals and objectives of this project. Following are presented the identified goals, objectives and finally the requirements of this project:

Goals:

G-I. Deploy WebComfort Applications in the context of a service

Objectives:

O-1. Allow the usage of existing WebComfort Applications on the WebC-SaaS Service context

Requisites:

- i.** Integrate WebC-SaaS with the WebComfort platform, while maintaining independency of other Applications;
- ii.** Maintain WebComfort independence of WebC-SaaS;
- iii.** Complement the WebComfort Kernel to include WebC-SaaS concepts (i.e. VariationPoints, ContextSettings), in order to provide Applications with new functionalities while maintaining independency of WebC-SaaS;

O-2. Allow the creation of various types of services

Requisites:

- iv.** Allow various service types to use the same WebComfort Module Definitions;
- v.** Differentiate Module functionality, depending on the level of service that is subscribed for a type of service;
- vi.** Offer different subscription plans for a service type;

O-3. Limit the area in which customers can access their services

Requisites:

- vii.** Implement customer sandboxes to define contexts where services can be used;
- viii.** Allow the subscription of various services on a single customer sandbox;
- ix.** Allow for sandbox customization, reusing WebComfort concepts like Roles, Tabs and Visual Themes.

G-II. Promote the WebComfort platform

Objectives:

O-4. Promote WebComfort services

Requisites:

- x.** Provide a mechanism for users to be able to preview a service without having to subscribe to it;
- xi.** Create a mechanism for users to easily refer existing services;
- xii.** Provide an easy way to access a service;

O-5. Allow subscriptions to have a trial period and/or to be payable

Requisites:

- xiii.** Support and implement various subscription methods;
- xiv.** Support paid subscriptions;
- xv.** Implement timed and pay-as-you-go subscriptions;

O-6. Guarantee minimum Service Level Agreements (SLAs) to customers

Requisites:

- xvi.** Present customers with information regarding the SLA before subscription;
- xvii.** Allow a customer to cancel a service subscription;
- xviii.** Log domain objects updates, in order to better diagnose, prevent and automatically recover from eventual issues;

G-III. Deploy the WebComfort platform in a Business context

Objectives:

O-7. Support Organizations on the WebC-SaaS context

Requisites:

- xix.** Support an Organization concept as a special Customer, with several users associated to it;
- xx.** Support the deployment of various SaaS instances for a single Organization context for load balancing;

O-8. Allow service subscriptions to be shared by various users on the context of an Organization

Requisites:

- xxi.** Update service subscriptions when new users are added or removed from Organizations;
- xxii.** Allow for only a subset of users to be responsible for the administration of Subscriptions;

G-IV. Deploy WebComfort over a distributed cloud computing environment, using a PaaS

Objectives:

O-9. Complement the WebComfort framework in order to support a PaaS service like Window Azure

Requisites:

- xxiii.** Implement a new Data Access library to work with PaaS data access methodologies on the cloud;
- xxiv.** Adapt the WebComfort framework to exploit Queues on the cloud, for implementing Event Publish/Subscription between WebComfort instances on the cloud;

O-10. Allow new WebComfort instance deployments

Requisites:

- xxv.** Implement automatic deployment of new WebComfort instances upon a service subscription;
- xxvi.** Update subscription states between WebComfort instances on the cloud;
- xxvii.** Allow for easy management of deployed instances on the cloud;
- xxviii.** Maintain functionality of a WebC-SaaS instance, independently of the cloud, so as to avoid unavailability of service;

Although some of these requirements have not yet been implemented on the current version of the project, all of these requirements have been taken into consideration throughout the conception of the WebC-SaaS infrastructure.

Because the WebC-SaaS project proposed several ambitious objectives to be implemented, the project had to be divided into milestones in order to produce actual results by the time of this project presentation. Because of this, an Iterative Software Development Process was adopted. This way, upon each milestone completion, it is possible to evaluate the evolution of the WebC-SaaS platform and to propose additional improvements to it.

3.3. Challenges

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.

Multi-tenancy

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.

Because multiple services would be able to be deployed by various customers on a single instance, it was necessary to create contexts to limit the area of action of customers and avoid illicit exploitation of private resources. To solve this issue, the Sandbox concept was introduced in the

WebC-SaaS Domain Model. Sandboxes are used to contain services and are limited to a single generic Customer, which can only access services' functionalities inside a Sandbox context. What's more, a Customer can only access services which are contained inside the particular Sandbox the customer is accessing.

Another possible implementation for multi-tenancy technologies is using PaaS services like Windows Azure to transparently implement this. However, these services were not available when this project's implementation started, so this made changing the Domain Model more complicated. What's more, this would create an enormous dependency of PaaS services, which was not ideal. For these reasons, the initiative to integrate WebC-SaaS with a PaaS service was discarded.

Context Settings and Variation Points

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.

To better understand this problem and its implementation, refer to section 4.5 of this document.

Integration with WebComfort

Because this project was to be developed as a WebComfort Toolkit, it was necessary to maintain WebComfort Applications' independence from it. For this reason, various approaches were necessary to be made, in order to maintain WebComfort independence from the WebC-SaaS Toolkit.

To separate the two namespaces, a WebComfort Extender was created to alter and complement the WebComfort platform functionality. This concept allowed, among other things, to handle platform and WebComfort events, useful in monitoring and altering the behavior of various objects, while maintaining independence between projects.

To better understand this problem and its implementation, refer to section 4.4 of this document.

Payment Methods

At the beginning of the development of this project, the WebComfort framework did not implement a Payment mechanism. Nonetheless, at the time of this project, this feature was being implemented by thesis student Wanderley Gonçalves which was working on the WebComfort e-Commerce toolkit. This toolkit focused on web payment methods, namely Credit Card and PayPal payments using Web Services. As such, in order to support payments on the WebC-SaaS project, it wasn't necessary to re-implement payment methods on the WebComfort platform, but rather integrate the project with the WebComfort e-Commerce toolkit to exploit its implemented payment methods.

To better understand this problem and its implementation, refer to section 4.5 of this document.

Instance Deployments and Maintenance

To implement instance deployment, multi-tenancy and load-balancing, it was considered using a PaaS service like Windows Azure, as these services implement those features and more.

To integrate this project with a PaaS service, it was necessary to explore how these services integrated with Web Applications. As such, various Platforms as a Service infrastructures marketed at the time were studied in more detail to explore its features and how they integrated with Web Applications like WebComfort. From this study, the Windows Azure and Amazon EC2 PaaS services were noticeably more interesting in comparison to other existing PaaS services. Both of these platforms integrated with Web Applications by providing a framework compatible with Microsoft .NET technologies in which WebComfort and WebC-SaaS are developed. What's more, these services offered a simple back-office which allowed administering instance deployments as well as their versions and statuses. Finally, these services offered attractive payment methods, compatible with SaaS payment methodologies like pay-as-you-go, which allow for better budget control.

In spite of this, at the time these services appeared, the Domain Model for this project was already prepared and implementation was advanced, so it was not possible to immediately integrate the developed WebC-SaaS model with a PaaS service. Nonetheless, efforts in that direction were made, to try to accomplish an integration with the Windows Azure platform. Alas, because these services were only beginning and still in Community Technology Preview (CTP), support for it was scarce and some required services were not available, so the integration was discarded.

3.4. Concepts and 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. For better understanding this Domain Model, the reader can access the DomainModel tab in the HtmlReport folder included in the CD that was annexed to this document.

Domain Packages

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 concepts;

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

Common package – Some classes that don't explicitly belong to any of the other packages, but are commonly used by classes from different packages. This package was created to group these concepts and avoid redundant concepts;

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

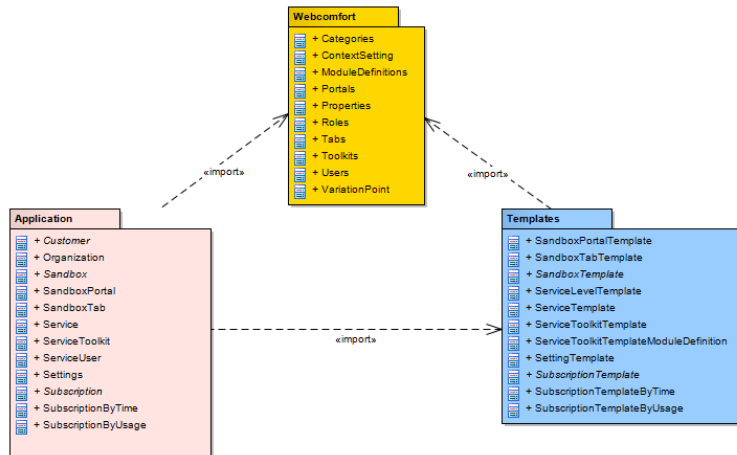


Fig. 8. Package dependency.

As it can be seen in this image, both the Application as well as the Template Packages depend on the WebComfort Package. On the other hand, the Application Package uses concepts of the Templates Package. This is due to the fact that some deployed concepts on the Application Package maintain a reference to the objects that were used as a template to originate these concepts.

Following are described the concepts of each package in more detail.

Template Package

Overview

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.

Next is presented an overview of the most relevant concepts of this Package:

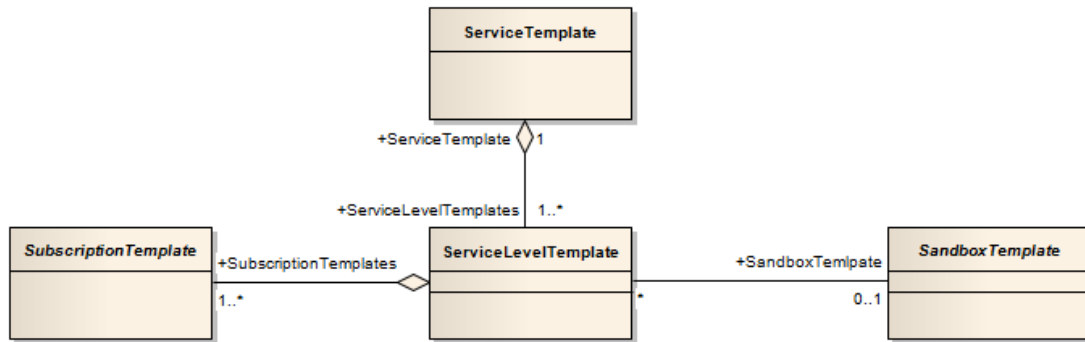


Fig. 9. Template Concepts Overview.

Note that some concepts appear in *Italics*, which means that those concepts represent abstract classes. This view reveals the relations between the main concepts of this Package.

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.

Sandbox Templates View

As previously described in the “Requirements” section of this work, the execution of services is required to be limited to a particular area. This area is named throughout this document as the service sandbox or microsite. ServiceLevelTemplates usually relate to a SandboxTemplate, in order to specify how modules should be displayed and the tabs that should be created upon a subscription. Nonetheless, services can be deployed over existing Sandboxes, so this makes the usage of SandboxTemplates facultative to some ServiceLevelTemplates.

Next is presented a view of the Sandbox Template concepts of the Template Package:

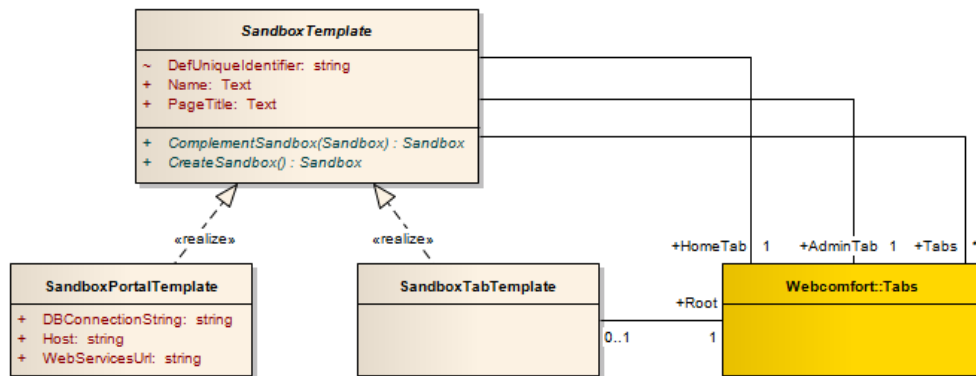


Fig. 10. Sandbox Templates View.

Note that SandboxTemplate is in *Italics* which means that it is an abstract class. Note also that although this concept is abstract, it has two distinct implementations: the SandboxPortalTemplate and the SandboxTabTemplate. The reason for developing two different implementations for this class had to do with the fact that requirements entailed that it should be possible to deploy Services in new or shared WebComfort instances. This way, SandboxPortalTemplates were included to generate new WebComfort Portals with all the necessary features installed, and SandboxTabTemplates were included to generate microsites on an existing WebComfort instance.

The SandboxTemplate has all the information relative to the construction of a Sandbox instance, like: Home, Admin and additional Tabs; Tabs’ organization and its contained modules and roles information; Visual Themes information; etc.

Implementing the SandboxTemplate class is the SandboxTabTemplate, which adds an additional “Root” field, which defines the TabID where new sandbox deployments using this Template should be installed. This type of SandboxTemplate creates a Tab hierarchy where the sandbox is confined.

Implementing the SandboxTemplate class is also the SandboxPortalTemplate, which adds additional information necessary to keep track of the deploying host. This type of SandboxTemplate creates a new instance of a WebComfort application which is used as a new sandbox.

Service Templates View

The Service Templates View is one of the most important views of the Domain Model. Here is where service types as well as their description, functionality, configuration and SLAs are defined to be able to promote and produce new services.

Next is presented a view of the Service Template concepts of the Template Package:

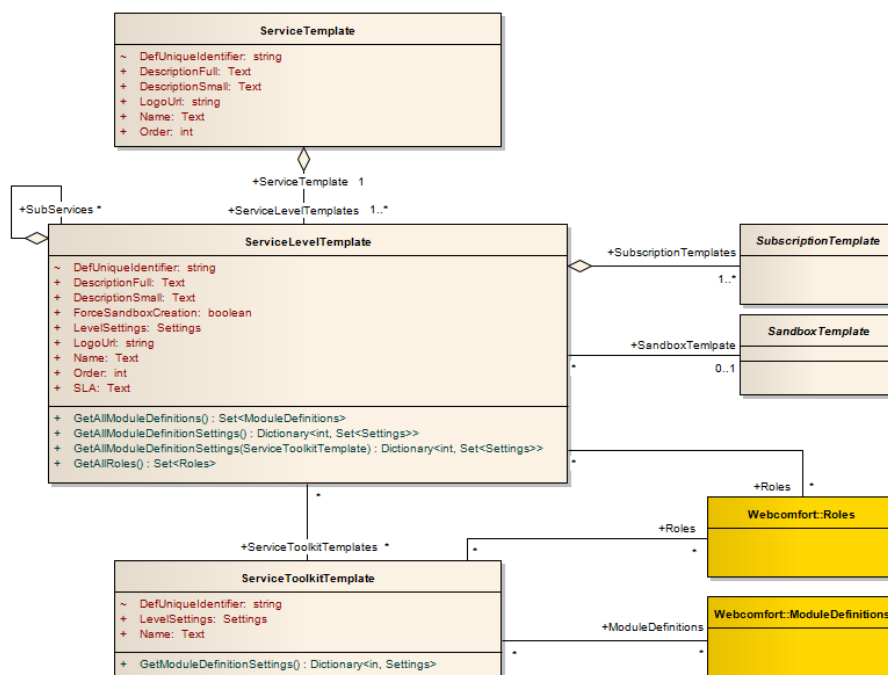


Fig. 11. Service Templates View.

The image above describes how the main concepts of the Service Template view are related to one another. Note that the ServiceTemplate works only as an aggregator of ServiceLevelTemplates. It is the ServiceLevelTemplate, along with the ServiceToolkitTemplate that define the functionality and configurability of the service.

In the ServiceToolkitTemplate in particular, it can be seen that this concept is used to define groups of ModuleDefinitions. This was needed, in order to allow the sharing of ModuleDefinitions by multiple service types, since WebComfort didn't implement a mechanism to group Module definitions in more than one group at a time. What's more, this ServiceToolkitTemplates were designed to be associated with a set of WebComfort Roles, in order to allow only a set of users to access the corresponding ServiceToolkits.

In studying the ServiceLevelTemplate, note that this concept has an association relation with ServiceToolkitTemplate. This is due to the fact that these two concepts can be considered to be at the same level of abstraction. Furthermore, notice that because ServiceToolkitTemplate definitions are independent from ServiceLevelTemplates, the ServiceToolkitTemplate can be shared by various ServiceLevelTemplate definitions. This is useful when implementing a series of service levels that have the same functionality (Service Toolkits), but differ in other aspects like configuration values. Note also that the ServiceLevelTemplate concept relates to the WebComfort Roles. This association is equivalent to that of the ServiceToolkitTemplate and the WebComfort Roles, previously explained. Another responsibility of the ServiceLevelTemplate concept is the definition of the Service Level Agreement document that is to be used as the guarantee for the contracted level of service.

Note also that the name, descriptions and logos included in the definition of the ServiceTemplate and ServiceLevelTemplate concepts are necessary to allow users to review a service type without having to subscribe to it.

Finally, note that both ServiceLevelTemplate as well as ServiceToolkitTemplate define LevelSettings. These settings are used to make use of the configuration properties of the WebC-SaaS platform (as discussed on section 3.3. – Context Settings and Variation Points). By defining these two levels of settings, it is possible to define one set of settings in the context of a ServiceToolkit, and another set of settings in the context of a Service that includes various ServiceToolkits. Nevertheless, know that ServiceLevelTemplate settings have precedence over other defined settings on the ServiceToolkitTemplate level. This means that for a particular setting which is defined on the ServiceToolkitTemplate and again in a ServiceLevelTemplate that includes that toolkit template; at a Service level, the setting will always be equal to the ServiceLevelTemplate value. Nonetheless, if a ServiceLevelTemplate does not define a setting which is defined in a containing ServiceToolkitTemplate, the service toolkit level settings are used. For a better comprehension of these settings templates, refer to the full Domain Model depicting the SettingTemplates concept, in the HTMLReport folder of the CD annexed to this document.

Subscription Templates View

As with other concepts in this Domain Model package, the SubscriptionTemplate concept is used to gather the data necessary to create a new Subscription in the Application package.

Next is presented a view of the Subscription Template concepts of the Template Package:

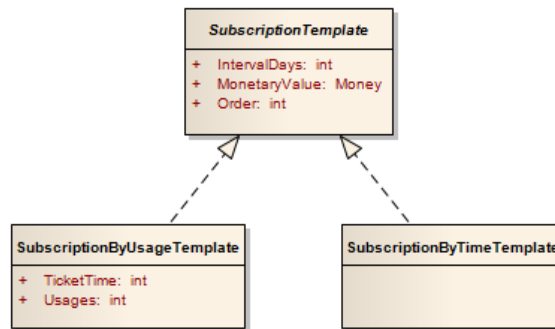


Fig. 12. Subscription Templates View.

The SubscriptionTemplate represents an abstract class, realized by two distinct implementations: the SubscriptionByUsageTemplate and the SubscriptionByTimeTemplate. This approach was taken to include one of the key concepts of SaaS applications on the project, which is the pay-as-you-go payment methodology. As such, all SubscriptionTemplate implementations were designed to support a periodic subscription, with a possible monetary value associated with it; but the SubscriptionByUsageTemplate was further developed to implement a pay-as-you-go payment methodology.

Application Package

Overview

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.

Next is presented an overview of the most relevant concepts of this Package.

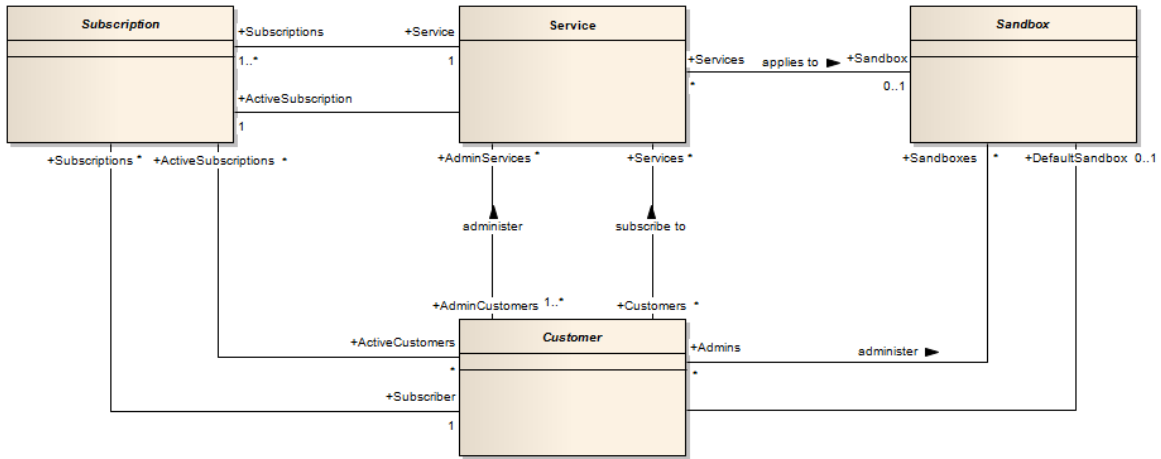


Fig. 13. Applicational Concepts Overview.

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.

Customers View

The Customers View represents how WebC-SaaS customers are defined.

Next is presented a view of the Customers' concepts of the Application Package:

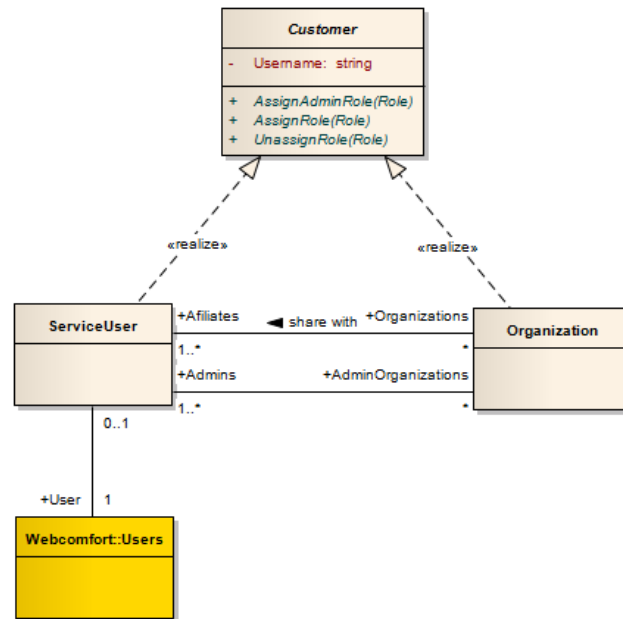


Fig. 14. Customers View.

As shown in the image above, the Customer concept is represented in *Italics*, which means it is an abstract class. This is due to the fact that when gathering the requirements for this project (see section 3.2), a requisite was identified that dictated that WebC-SaaS should support Organizations. For this reason, a special Customer type should exist to model this concept. This way, two distinct realizations of the Customer class were implemented. The Customer class defines a generic interface to interact with the WebC-SaaS platform and all its concepts. It can also define the necessary information to carry out payments.

The ServiceUser class implements a standard customer, with an associated user account in the WebComfort platform. This customer definition was created to complement the WebComfort User concept while serving as an interface to interact with all WebC-SaaS concepts. This type of object is automatically and invisibly created for an authenticated WebComfort User.

The Organization class is another implementation of the Customer concept, and is capable to assemble all the information of an Organization. This concept is not directly associated to a WebComfort User, but relates to multiple ServiceUser customers that are allowed to access and/or administrate the Organizations' subscriptions as well as configure other properties. A ServiceUser customer is allowed to be part of various Organizations. The Organization concept, nonetheless, must be created manually by a customer.

Sandboxes View

The Sandboxes View models the sandbox concept and its implementation types.

Next is presented a view of the Sandboxes' related concepts of the Application Package:

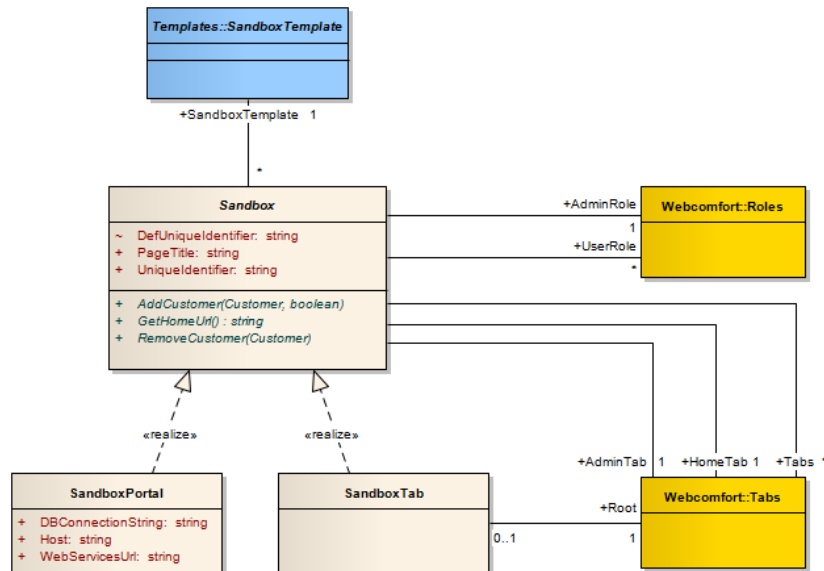


Fig. 15. Sandboxes View.

The Sandbox concept is an abstract class that represents the area in which services can be executed. There are two possible implementations for this class, which are the **SandboxPortal** and the **SandboxTab**. All of these classes are generated from a particular **SandboxTemplate** definition. For this reason, a reference to the originating **SandboxTemplate** is saved on the **Sandbox** class.

The **SandboxPortal** class represents an implementation of the **Sandbox** abstract class, and defines a **Sandbox** that is capable of managing services on a single-tenant WebComfort instance. For this reason, the **SandboxPortal** must define the necessary references to identify and communicate with the originating WebComfort instance. Objects of this type are created and complemented from **SandboxPortalTemplate** definitions.

The **SandboxTab** class represents an implementation of the **Sandbox** abstract class, and defines a **Sandbox** that is capable of managing services on a shared WebComfort instance. These **Sandbox** types are created within a shared WebComfort instance, using a **Tab** hierarchy to define the sandbox context. Objects of this type are created and complemented from **SandboxTabTemplate** definitions.

Services View

Next is presented a view of the Services' concepts of the Application Package:

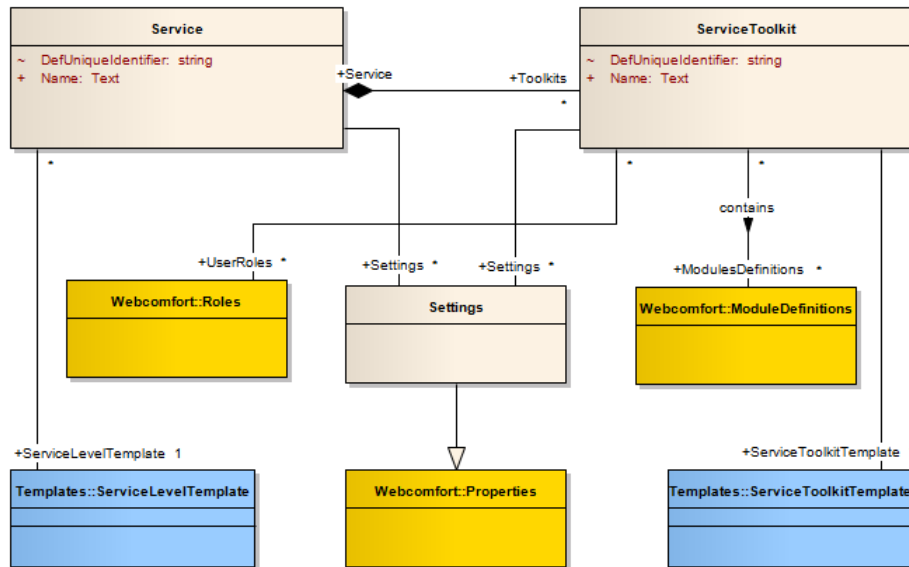


Fig. 16. Services View.

The Service class represents a WebC-SaaS service. This concept is used throughout this platform to capture the module definitions, settings and roles a service defines. This class is generated from a particular ServiceTemplate object. In order to be able to track this relation in the future, a reference to this object is maintained in the Service class.

To delineate which module definitions a service provides, an association with the ServiceToolkit class exists. This concept is used throughout the WebC-SaaS platform to allow service customers to add/remove and configure various Module Definitions on the service Sandbox. This class is generated from a particular ServiceToolkitTemplate definition. Because of this, it is useful to keep a reference to the originating ServiceToolkitTemplate object.

Subscriptions View

Next is presented a view that comprehends the Subscriptions' concepts on the Application Package:

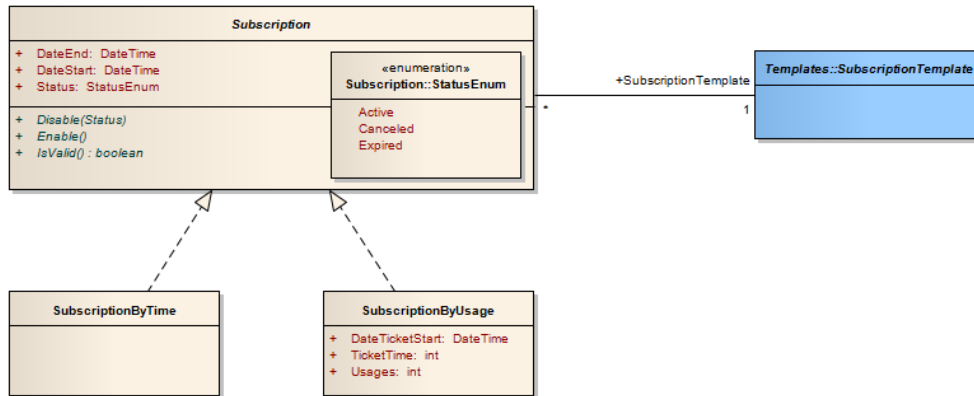


Fig. 17. Subscriptions View.

The Subscription concept comprehends the information about a service subscription, and the mechanisms necessary to validate these when customers try to access a service. Typically Subscriptions have a maximum duration associated to it. Because it was decided that there should be various types of Subscription, the Subscription concept was designed to be an abstract class. Implementing this class are the SubscriptionByTime and SubscriptionByUsage types.

The SubscriptionByTime class defines a periodic Subscription, limited to a period of time. These Subscription types define the date in which they were created and the date until when they will no longer be valid.

The SubscriptionByUsage class defines a Subscription limited to a number of usages, in order to help implement the pay-as-you-go payment mechanism. These Subscription types are based on the usage of tickets to determine when a subscription is valid. The SubscriptionByUsage class includes the number of tickets a subscription has available (Usages), the last time a ticket was used (DateTicketStart) and the duration of a ticket (TicketTime) in seconds. The way the validation of this type of subscription works is that it verifies the last time a ticket was used, and checks to see if that ticket is still valid considering the duration of a ticket. When the ticket is no longer valid, then another ticket must be used. When there are no more tickets available to be used, a subscription ceases to be valid.

WebComfort Package

In this package is only referred the ContextSetting and the VariationPoint concepts, which were the only additions to the WebComfort model. This is to do with the fact that implementing ContextSettings, WebComfort Applications were required to be aware of the existence of these concepts. To maintain Application's independence of the WebC-SaaS toolkit, these concepts were added to the WebComfort Kernel namespace.

Context Settings View

As seen in the Challenges section of this chapter, the VariationPoint and ContextSetting concepts are needed to define settings for different contexts.

Next is presented a view that comprehends the ContextSetting and VariationPoint concepts on the WebComfort Package:

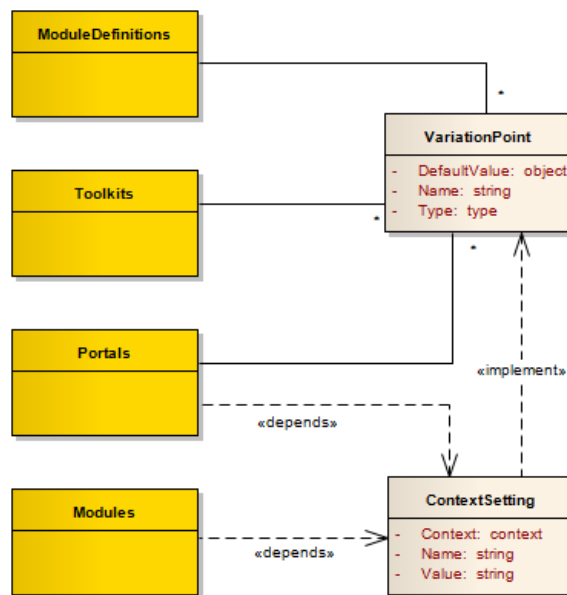


Fig. 18. Context Settings View.

The VariationPoint type is necessary to declare that a setting will be programmatically used in an application. This declaration includes the setting name which is going to be searched for by the application, the type of value of the setting (i.e. int, enum, array, etc.) and a default value for that value. This concept can be stated by ModuleDefinitions, Toolkits or Portals.

On the other hand, the ContextSetting is needed to store the actual setting that is to be used by applications. This concept gathers information about a setting's name, the context that defines a particular setting, and the value of the setting. Objects of this type implement settings that derive from a VariationPoint, but can change its value. Context information is gathered to support context searching.

4. WebC-SaaS – 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.

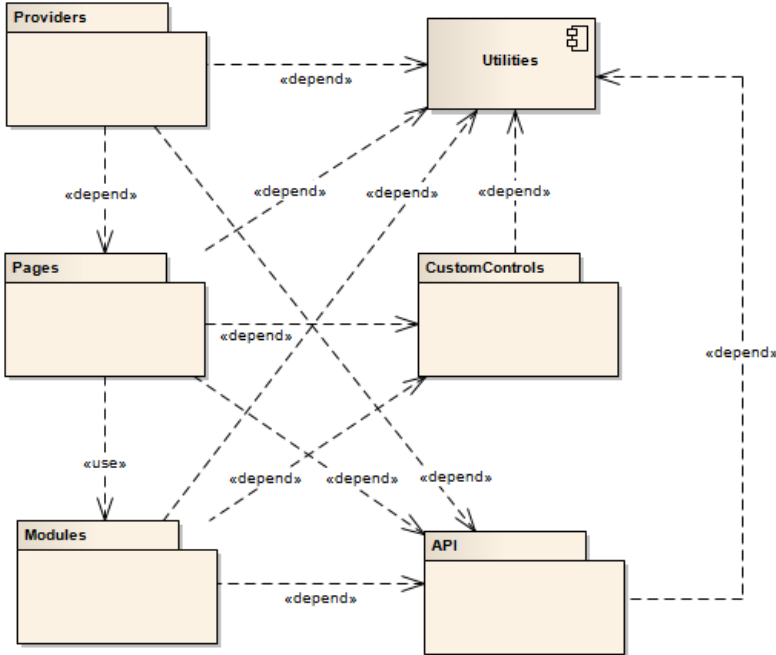


Fig. 19. WebC-SaaS components.

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.

Note that this information can be consulted at any time in more detail accessing the HtmlReport folder included on the CD annexed to this document.

4.1. Integration

The WebC-SaaS project was conceived to be organized as a standard WebComfort Toolkit, composed of a set of WebComfort Modules and Pages, as well as a set of functional components including Providers and APIs. Because of this, the WebC-SaaS Toolkit can easily be integrated in the WebComfort architecture.



4.2. 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.

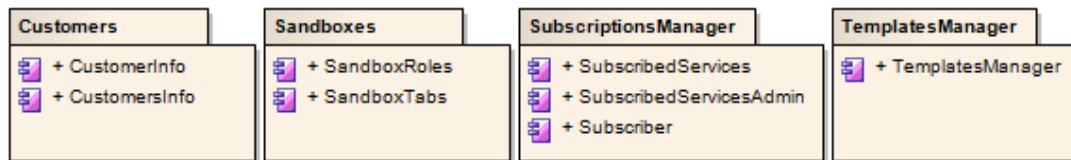


Fig. 21. WebC-SaaS modules.

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.



Fig. 22. WebC-SaaS Module examples.

In the image above are presented as an example the Subscriber, TemplatesManager, SubscribedServices and SubscribedServicesAdmin modules, respectively.

4.3. 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.

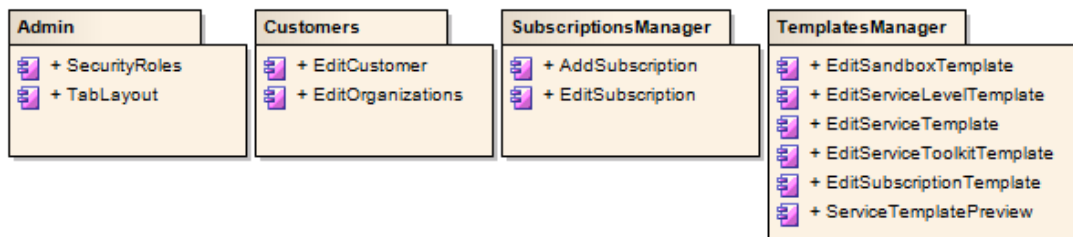


Fig. 23. WebC-SaaS pages.

Templates Management

Most of the pages defined in this project exist to support the creation and definition of ServiceTemplates, ServiceLevelTemplates and all other subsequently necessary templates. Next is presented the typical workflow in defining these templates.

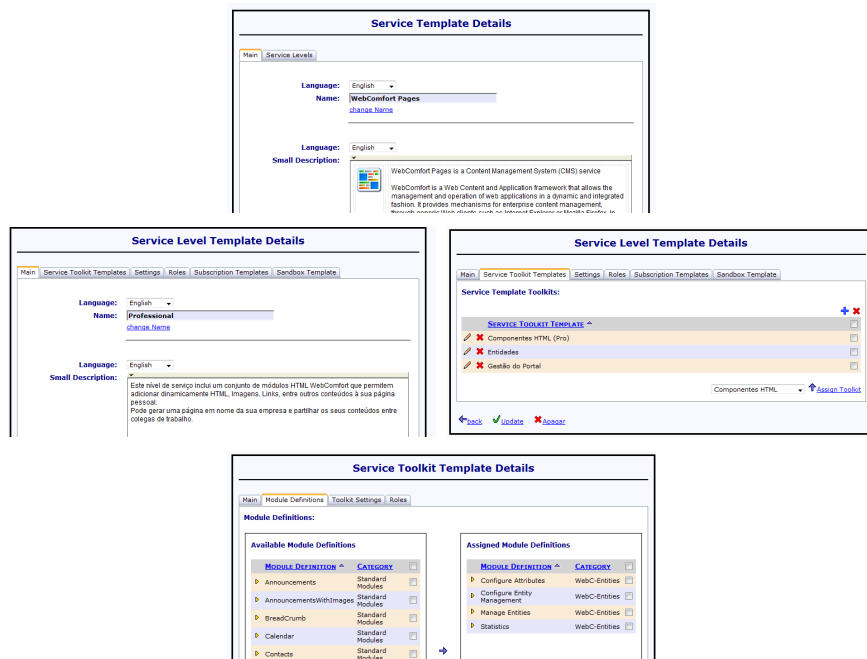


Fig. 24. WebC-SaaS templates definition.

As referred in the previous section, ServiceTemplates can be created, edited and removed by using the ServiceTemplatesManager module. Nonetheless, defining a ServiceTemplate and its ServiceLevelTemplates, it is necessary to access the EditServiceTemplate page, by clicking the Edit button of the referred module. Once in this page, an administrator is presented with two panels: Main and Service Levels. In the Main panel, the administrator is able to enter the information relative to the presentation of the ServiceTemplate. This is used to promote a service type, and allow a user to preview the service's main features before subscribing to it. On the other hand, in the Service Levels panel of this page, a list with the ServiceLevelTemplates associated to this ServiceTemplate is presented. Here it is possible to add, delete and edit these Service Levels. Clicking the Add or Edit button in this section, redirects the administrator to the EditServiceLevelTemplate.

In the EditServiceLevelTemplate page, an administrator is shown six panels: Main, Service Toolkit Templates, Settings, Roles, Subscription Templates and Sandbox Template. Here, the administrator is able to: define the presentation and SLA in the Main panel; assign or create

ServiceToolkitTemplates on the Service Toolkit Templates panel; define service-wide settings for a level of service; define which roles should be associated with the originated service; what subscription templates can exist for the level of service; and create, assign and configure the behavior for SandboxTempates.

Another crucial step in defining a subscribable service type is defining a SubscriptionTemplate for the ServiceLevelTemplate. To assist in creating these subscription templates there is the EditSubscriptionTemplate page. Furthermore, to define functionality for a ServiceLevelTemplate, the page EditServiceToolkitTemplate was created. Finally, to help create new SandboxTemplates, the EditSandboxTemplate page was created. All of these pages are accessible when trying to add or edit each of these concepts on the EditServiceLevelTemplate page.

Subscription process

After clicking a service subscription button customers are redirected to the AddSubscription Page. This page is responsible for gathering the final information necessary to complete a subscription. Next is presented the typical workflow during a subscription of a WebC-SaaS service.

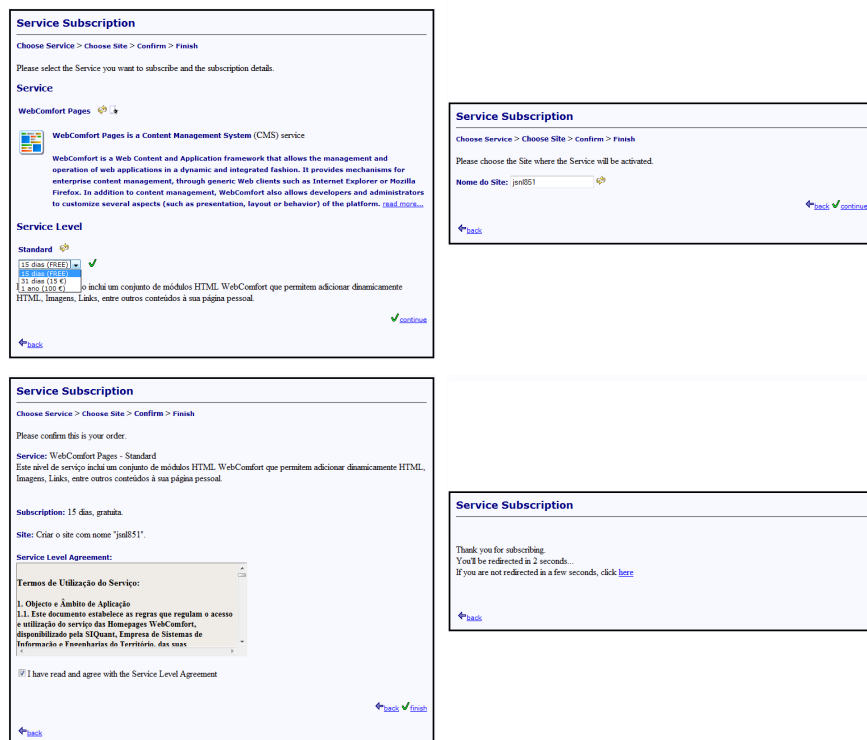


Fig. 25. WebC-SaaS subscription workflow.

As it can be seen, this page workflow has three steps before the subscription can be created.

In the first step of the subscription process, the user can view in more detail the information about the service subscription it is about to do. What's more, here the user can switch between services subscription options to find the solution it wants. Moreover, it is also in this phase that the user identifies the subscriber of the service. In most cases, the subscriber will be the customer associated with the WebComfort user account, but in some occasions – namely when a user belongs to an Organization – it can be helpful to be able to subscribe to a service in the name of an Organization. Note that, if a WebC-SaaS Customer account has never been created for a user, this will be done automatically, and the customer will be used as the default subscriber. Finally, it is in this step that the user chooses the subscription type he wants associated to the service.

The second step of this process is to identify the sandbox where the service is to be deployed. Because Service types don't have to include a SandboxTemplate definition, here it is possible to select a sandbox if the subscriber already has an associated microsite. If not, the Service has to define a SandboxTemplate, and the user has to designate the name for the new microsite.

As a final step of this process, it is necessary to review the Subscription order, and to confirm it. Here, if the Service type being subscribed to has a Service Level Agreement associated to it, the user is obligated to comply with it, by checking a checkbox.

Upon completion of this process, the user is redirected to the newly subscribed service Home Page.

An alternative workflow for a service subscription may include a payment process if the selected Subscription type is paid. In this case, the workflow is the same as the previously described, but after completing the confirmation step – which is usually the last – of the subscription process, the workflow is redirected to the WebC-eCommerce payment workflow. Here, the user introduces the payment method and the associated data to proceed the payment, and confirms it. Upon successful completion of this payment, the user is redirected to a receipt page with the information concerning the payment, and the service is automatically created. Finally, because the user is not redirected to the new service's Home Page, the user has the option of going back to the subscription page, where the subscription workflow is concluded, and the user is redirected to the newly created service Home Page.

Sandboxes administration

In using sandboxes, various approaches for configuring sandbox tabs and its components were considered. It was decided that the best choice to allow a user to configure these components would be to reuse the concepts employed in the previous WebComfort system. This way, an “Edit tab” button was added to the top right corner of each Sandbox page, replicating the “Edit tab” button concept of the previous WebComfort system. When a user selected this button, it would be redirected to a page containing similar options to that of the configuration page of the previous WebComfort system, but with fewer options available to the user.

To provide an even more familiar use of the WebC-SaaS sandboxes, the users were provided with a mechanism that allowed defining a default Home Page to be accessed when the user first authenticated on the site, or by clicking in the “Home” button on the links section of the banner.

4.4. Providers

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.

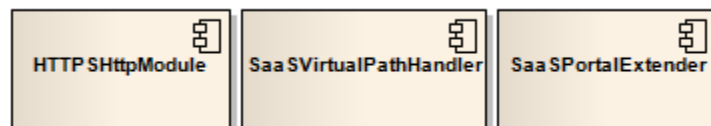


Fig. 26. WebC-SaaS Providers.

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.

4.5. Other Technical aspects

Some technical details could not be explained in the previous parts of this Implementation section of the document. In this chapter, the remaining relevant technical aspects are presented.

Context Settings and Variation Points

As seen in section 3.3, one of the challenges of this project is implementing context settings to support configurability of SaaS applications. As described in that section, two concepts were needed to implement this: VariationPoint and ContextSetting. Next is explained in more detail how these concepts were implemented.

Because Application Modules only have to define their VariationPoints once, it was decided that the association with these objects with the respective Application Modules would be best made during the installation of the Modules. As such, it was decided that Application Modules would have to include the VariationPoints' definitions to the containing Toolkit's definition, before it being installed. Afterwards, when these Toolkits were to be installed, it would be possible to obtain all VariationPoints' definitions and install them accordingly, associating these to the Module Definitions. This association would then be useful to alert the WebComfort platform that certain settings should be configured, so that afterwards Modules could retrieve the VariationPoint's associated settings.

Alternatively, for debugging purposes and version control, another method of adding Variation Points was created. This time, the process was not meant to be automatic, as Variation Points were added and/or removed manually. To implement this feature, a new WebComfort Admin Module was added, to allow to add/remove/search for Variation Points and its associated Module Definitions.

The other substantial challenge of implementing WebComfort configurability was to develop Settings' storage and recovery for different contexts. As an example, consider that a document management Module Definition has defined the VariationPoint "NUM_DOCS_MAX" with a default value of "1000" and that this setting is used in *run-time*. Two services are subscribed by two customers: "Service1" by "Cust1" and "Service2" by "Cust2". Both services provide access to a shared document management Module "Module1". The first service had defined that the setting "NUM_DOCS_MAX" would have the value "100" and the second service didn't define a setting for the "NUM_DOCS_MAX" VariationPoint. In this scenario, customer "Cust1" accesses "Module1" and the

module retrieves a context setting of “100”. On the other hand, customer “Cust2” can also access “Module1”, but this time the module retrieves a context setting of “1000” because in this context no value was defined for “NUM_DOCS_MAX”, but in a parent context level there is a setting value for that VariationPoint, which is “1000”.

It is clear that different contexts can exist on the WebComfort and WebC-SaaS platforms, depending on various aspects that define an execution context (e.g. which Tab, User, Service, Sandbox, etc. is being used). What it isn't clear is how to assemble and decompose the levels of these contexts; which context has precedence over the other; and what rules have to be verified in order to identify the correct value for a setting in a particular context.

In order to implement this intricate context settings scheme, it was employed a dynamic mechanism for pushing/popping contexts to an execution stack of contexts, which is populated every time a setting is sought (whether for storage or retrieval). What's more, this context scheme has associated events to indicate when a stack is being populated, so this makes possible to programmatically change the behavior of a stack filling, by allowing different implementations to add/remove new contexts when a “stack context populate” event is raised. Finally, there is a need to clarify the rules for deciding which setting should be chosen when settings are only found on a parent context, and not on the original context. To resolve this, a dynamic function exists with simple rules to decide which context should have precedence over the other. This function can then be altered by other context implementations, in order to consider different decision rules in the process.

Payment Methods

As seen in section 3.3, integration with another WebComfort Toolkit was necessary to add payment support to the WebC-SaaS platform. This task required a closer collaboration between the two parts involved, to discuss how the integration should be done.

To implement this feature, the WebComfort e-Commerce toolkit introduced the PaymentOrder class to represent a payment order. This class assembled all the information necessary to produce a payment over the WebComfort platform. Once created an object of this type, it would then be necessary to redirect to the WebComfort e-Commerce payment workflow. On this context, customers would be able to see the associated payment order as well as provide with the information necessary to proceed with the payment. Upon successful payment completion, the WebComfort e-Commerce toolkit would launch an event to signal that a payment order was completed. Finally, WebC-SaaS would capture this signal, confirm that the payment was successful and not yet used, and create the service subscription using the Payment Order information, guaranteeing this way that services are always created upon a successful payment.

5. 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.

Following are described the two test cases for this validation.

5.1. Pages Services

As described above, one of the test cases consisted in creating a service, capable of assembling various standard web content management WebComfort modules. To do this, the project WebComfort Pages was created, and an instance of WebComfort was deployed at the location <http://pages.webcomfort.org/>.

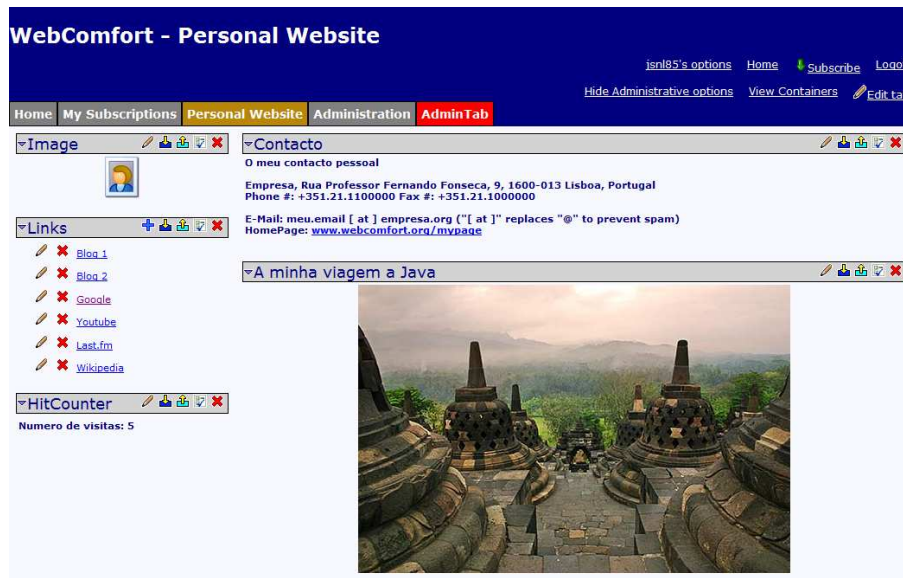


Fig. 27. Homepage of a deployed Personal WebSite service.

This service is particularly interesting to new users that want to explore the WebComfort platform, whether on a free or a trial basis. What's more, this project allows WebComfort developers to easily test new Modules with customers on a production environment, and eventually develop and distribute new integration updates to the WebC-SaaS project.

5.2. WebTrails Services

As described above, another of the test cases consisted in creating a service, capable of integrating the WebTrails application. To accomplish this, it was necessary to configure and alter a WebTrails instance to implement Context Settings as well as Variation Points. Since these concepts were included in the WebComfort Kernel namespace, it was possible to rapidly integrate the two projects. As a result, the project WebComfort WebTrails was updated to a new version, but at the time of this presentation, an instance of this solution was not put in production.

Portal de Percursos e Interpretação

Home Links About Home Login Sign Up

Portal Tree

- Portal de Percursos e Interpretação
 - Home
 - Links
 - About

You are the visitor number 20484.

English Português

Introduction









WebTrails is a Web-based application which supports the visualization and autonomous management of multimedia content by different stakeholders (eg, Municipalities, Natural Parks, Tourism Associations) regarding information of Trails and Interpretation. In the end, these contents can be exported in an open XML-based format, updated in the PDA, and finally viewed and used again as part of MobileTrails solution.

MobileTrails is a software solution for PDA or PocketPC to support and complement the experience of sightseeing, walking in the parks, urban centres and historic cities. Taking advantage of the integration of geographic maps and additional location-based equipment (GPS equipment).

The project has currently the development of "Trails and Interpretation" for the National Park of Peneda Gerês (PNPG) and to Sintra-Cascais Natural Park.

For more information contact mail@sigquant.pt.

List of Contexts

	Parque Nacional da Peneda-Gerês (PNPG)	 10	 266	 116
	Sintra-Cascais Natural Park	 1	 13	 13

Subscriber

Check out our services in the list below.
Hint: You can preview any service by clicking it.





WebTrails Partner	Basic	 Subscribe
	Standard	 Subscribe
	Prime	 Subscribe
WebTrails Visitor	Standard	 Subscribe

Fig. 28. WebTrails application, integrated with WebC-SaaS Services.

6. Conclusion

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 innumerable 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.

6.1. Future Work

As a final note, some suggestions of improvements to this project's work are presented below. Due to the fact that this project was somewhat ambitious in its goals and objectives, some features weren't able to be developed in time of this presentation. For this reason, some of these suggestions can comprise in small adjustments to the developed work.

Instance deployment

One feature that wasn't implemented in time for this presentation was instance deployment, which consists in automatically setting up new instances of the WebC-SaaS platform, along with all the necessary WebComfort Toolkits used by subscribed services, onto the Internet. This feature, along with multi-tenancy and load balancing were considered to be implemented recurring to PaaS services, but due to difficulties at the time integrating with the service provider, that approach was discarded. Nevertheless, this feature is related to several of the WebC-SaaS requirements, so this makes it pertinent to the project's success.

Subscription to already deployed services

Although the project, as it presented, already supports for services to be deployed on an existing sandbox, some features in this circumstances are still lacking. Specifically, two features were identified as more relevant in this situation: Sandbox complementation and context Subscriptions.

Context Subscriptions refers to subscriptions which are executed over existing deployed services. As it is implemented, subscriptions to services can only be made at a Portal level. Nonetheless, for some types of services, it can be pertinent to define sub-types of service. This is especially interesting when service subscribers are concerned in sharing information with other customers, but don't want to manually define permissions to let customers access its contents. In this case, it can be pertinent to implement a subscription module to define a sub-type of service, of the existing service.

Another feature that could be implemented in future versions is sandbox complementation, which refers to integrating various SandboxTemplates in a single Sandbox. When subscribing to various services on the WebC-SaaS platform, various sandbox templates can be present. Nonetheless, as of this version of the project, only one Sandbox Template can be used to generate a Sandbox. For this reason, it is not possible to merge the disposition of other SandboxTemplates in an existing Sandbox, so this can make some features become hidden initially, as subsequent Modules included in SandboxTemplates are not added to the existing Sandbox.

Integration with the WebComfort kernel

Although this project was developed as an independent Toolkit to WebComfort, the work here implemented can easily be integrated into it. Although this can possibly restrict WebComfort in terms of implementing alternative service subscription models, this integration can yield significant efficiency and consistency improvements to the platform.

As a WebComfort toolkit, WebC-SaaS has to rely on providers like the WebComfort Extender class to implement extensions to the platform. If this work was integrated with the WebComfort platform, these extensions could be implemented exactly where and when they were needed, making the system more efficient and essentially more consistent. Moreover, this would allow employing additional technologies which would be even more advantageous.

References

1. Dimitrios Georgakopoulos, Norbert Ritter, Boualem Benatallah, Christian Zirpins, George Feuerlicht, Marten Schoenherr, Hamid R. Motahari-Nezhad.: Service-oriented Computing. Springer (2006)
2. Alexander Factor.: Analyzing Application Service Providers. Prentice Hall (2001)
3. Nicolas Gold, Claire Knight, Andrew Mohan, Malcolm Munro.: Understanding Service-Oriented Software. IEEE Software (2004)
4. Prashant Palvia, Aaron Patula and John Nosek.: Problems and Issues in Application Software Maintenance. Journal Of Information Technology Management (1995)
5. Michael Alan Smith and Ram L. Kumar.: A theory of application service provider (ASP) use from a client perspective. Elsevier (2004)
6. Bandula Jayatilaka, Andrew Schwarz and Rudolf Hirschheim.: Determinants of ASP Choice: an Integrated Perspective. Hawaii International Conference on System Sciences (2002)
7. SIIA.: Software as a Service: Strategic Backgrounder. Software & Information Industry Association (SIIA) (1999)
8. Keith Bennett, Paul Layzell, David Budgen, Pearl Brereton, Linda Macaulay and Malcolm Munro.: Service-Based Software: The Future for Flexible Software. Bennet et al. (1999)
9. Mark Turner.: Turning Software into a Service. Keele University paper (2006)
10. WebComfort (<http://www.webcomfort.org/Portal>)
11. Alberto Rodrigues Silva, João Saraiva.: The WebComfort Framework: An Extensible Platform for the Development of Web Applications. 34th EUROMICRO Conference on Software Engineering and Advanced Applications paper (2008)
12. SIQuant (<http://www.siquant.pt/>)
13. Gabriel Coimbra.: Software as a Service: As múltiplas dimensões do SaaS. Microsoft Solutions Day conference (2008)
14. D. Chappell.: Introducing the Azure Services Platform - an early look at Windows Azure, .NET services, SQL services and Live services (October 2008) (http://download.microsoft.com/download/e/4/3/e43bb484-3b52-4fa8-a9f9-ec60a32954bc/Azure_Services_Platform.pdf)
15. H. Erdogmus.: Cloud computing: Does Nirvana hide behind the Nebula? IEEE Software (2009)
16. M. Fitzgerald.: When the forecast calls for clouds. Boston University (2009)
17. B. Hayes.: Cloud computing. ACM (2008)
18. J. N. Hoover.: A stake in the cloud. InformationWeek (2008)