Software-as-a-service (SaaS): Considerations and implications for SaaS customers

by
Jacobus Frederick Dippenaar
Student number 13063324

Dissertation presented in partial fulfilment of the requirements for the degree Master of Accounting in Computer Auditing at the University of Stellenbosch

Supervisor: Prof Rika Butler
Faculty of Economic and Management Sciences
Department of Accountancy

December 2008

Verklaring

Deur hierdie werkstuk elektronies in te lewer, verklaar ek dat die geheel van die werk hierin vervat, my eie, oorspronklike werk is, dat ek die outeursregeienaar daarvan is (behalwe tot die mate uitdruklik anders aangedui) en dat ek dit nie vantevore, in die geheel of gedeeltelik, ter verkryging van enige kwalifikasie aangebied het nie.

Datum: 29 Oktober 2008

……

JF Dippenaar

Studente nommer 13063324

Abstract

Software-as-a-Service (“SaaS”) is a software delivery model whereby software applications, such as Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), Human Resource administration and payroll and Procurement, are hosted centrally by various service providers at their premises. These hosted applications can be delivered to multiple service customers via an existing Internet connection, with a browser based front-end, or via a thin client system.

As all hardware and support services are provided by the service provider, operational costs for customers are reduced in comparison to a traditional, in-house supported software application. Traditional cost estimates for the deployment of on-premise software applications exclude personnel costs that are needed for ongoing support and maintenance. Depending on certain variables, such as the application involved, these costs can vary between 50% and 85% of the total cost of ownership of the application. Additionally, the cost of maintenance, periodic upgrades and continued support, on an annual basis, can be up to four times the initial cost of purchasing the application.

From the perspective of the service customer the most difficult part of determining whether to move to SaaS is the total cost of ownership (“TCO”) calculation. This is due to the fact that the TCO does not merely include the cost of new licences, but also the careful consideration of certain variables, before a potential service customer can make a decision regarding a potential move to SaaS. These variables include considerations such as:

- Physical and logical communication interfaces;
- User requirements;
- Security and privacy of information and data;
- Customisability;
- Availability of services and data;
- Service levels;
- Data ownership; and
- Integration with existing systems.

Presently no comprehensive framework exists that sets out the various aspects to be considered by a user company when determining whether to adopt SaaS, or not. The purpose of this study is to provide a comprehensive framework of considerations relating to the adoption of SaaS by user companies in the form of a Total Cost of Ownership calculation. The framework was compiled after considering the variables that would influence the decision to move to SaaS, the impact of these variables on the potential SaaS customer and the costs associated with each of the above aspects.

The framework compiled can assist potential SaaS customers in the decision to adopt SaaS. In addition, this table of considerations can also be used by the potential SaaS vendor in determining the viability of their SaaS offering, when compared to an equivalent, on-premise based software solution.

Uittreksel

"Sagteware as 'n Diens" is 'n sagteware-leweringsmodel waardeur sagteware toepassings soos Onderneming Hulpbron Beplanning, Kliënte Verhoudings Hantering, Menslike Hulpbronbestuur en Betaalstelsel en Aankopebestuur, sentraal by verskeie diensverskaffers op hul persele gehuisves word. Hierdie toepassings word deur middel van bestaande Internet konneksies aan verskeie klante beskikbaar gestel. Dit kan geskied deur middel van 'n web-blaaier ("browser based front-end") of deur 'n dun kliente stelsel ("thin client system").

Aangesien alle hardeware en ondersteuningsdienste deur die diensverskaffer gelewer word, is operasionele kostes vir klante laer as in vergelyking met 'n tradisionele intern ondersteunde sagteware toepassing. Tradisionele koste beramings vir die implementasie van interne sagteware toepassings sluit personeelkostes nodig vir deurlopende ondersteuning en instandhouding uit. Afhangend van sekere veranderlikes kan hierdie kostes tussen 50% en 85% van die totale koste van eienaarskap van die toepassing wees. Bykomend hiertoe kan die koste van instandhouding, opgradering en deurlopende ondersteuning tot vier keer die aanvanklike koste beloop.

Alhoewel Sagteware as 'n Diens verskeie voordele inhou vir die klant, is daar ook verskeie oorwegings wat 'n potensiele Sagteware as 'n Diens klant in ag moet neem voordat die besluit geneem word om na hierdie model te beweeg. Dit sluit die volgende aspekte in:

- Fisiese- en logiese kommunikasie koppelings;
- Gebruiker vereistes;
- Sekuriteit en privaatheid van inligting en data;
- Pasmaakbaarheid;
- Beskikbaarheid van dienste en data;
- Diensvlakke;
- Eienaarskap van data; en
- Integrasie met bestaande stelsels.

Die doel van hierdie studie is om 'n omvattende lys van oorwegings, wat verband hou met die besluit van 'n potensiele Sagteware as 'n Diens klant om te beweeg na 'n Sagteware as 'n Diens model, uiteen te sit in die formaat van 'n Totale Koste van Eienaarskap ("TKE") berekening. Huidiglik bestaan geen soortgelyke raamwerk nie.

Die raamwerk is opgestel nadat die aspekte hier bo gelys, die impak van hierdie aspekte op die potensiele Sagteware as ‘n Diens klant en die koste betrokke by elkeen van die aspekte, in ag geneem is.

Hierdie raamwerk kan dan gebruik word deur die potensiele Sagteware as ‘n Diens klant in die besluit om te beweeg na ‘n Sagteware as ‘n Diens model. Addisioneel tot dit, kan die raamwerk ook deur ‘n potensiele Sagteware as ‘n Diens verskaffer gebruik word om die lewesvatbaarheid van hul Sagteware as ‘n Diens toepassing te bepaal, in vergelyking met ‘n ekwivalente, op-perseel sagteware toepassing.

Index

Listing of figures

Listing of tables

Chapter 1 – Introduction

1. Background

The main objective of for-profit companies should be to increase revenue, decrease costs and thus, increase bottom-line profits and shareholder value (Sysmans, 2006). One way to achieve this is to use all available resources efficiently and effectively to support the primary revenue generating activities of the entity. This implicates that any other activities, that do not directly generate revenue, are secondary. As secondary functions do not contribute directly to the bottom-line profits, the best manner in which to address these functions is to perform them in a standard, efficient and effective manner (Sysmans, 2006). Certain secondary activities therefore lend themselves toward outsourcing, rather than being performed in-house. In the context of software applications, outsourcing refers to "the transfer of components or large segments of a company's internal IT infrastructure, staff, processes or applications to an external resource…" (Software & Information Industry Association, 2001)

When an entity (service customer) outsources a secondary activity, their secondary activity becomes another company's (service provider's) primary revenue generating activity (Sysmans, 2006). The service provider is incentivised to deliver the most effective and efficient service to the customer, employ specialised skilled personnel and put in place the necessary infrastructure to deliver the service. The benefit to the customer is therefore clear: they receive the most effective and efficient service whilst valuable resources are freed-up to focus on the primary activities of the business that increase bottom-line profits and increase shareholder value.

Many companies see IT functions as secondary activities of the business (Businessweek, 2006) and therefore suitable for possible outsourcing as opposed to delivering IT functions through traditional on-premise software installations. This is mainly due to the fact that most software applications do not differentiate a company from its competitors and therefore does not contribute to the company's primary business objective (Sysmans, 2006). Furthermore, companies spend on average four times the cost of the initial purchase of an application on the support and maintenance thereof. As a result up to 75% of the IT budget is allocated to the maintenance of hardware and software infrastructure.\ (Sysmans, 2006) Whereas with traditional on-premise software installations, the bulk of

the IT budget of companies is allocated to support and maintenance of hardware and to professional services, leaving only a small portion of the budget for software applications (Carraro & Chong, 2006a).

Software-as-a-Service ("SaaS") provides a technique for outsourcing the IT activities of a company. SaaS is a software delivery model, by which a web-native software application is hosted centrally by a SaaS vendor (service provider), access to the application is delivered to multiple customers over a network, such as the Internet, via a web-browser and fees are normally charged on a recurring or usage basis. (Software & Information Industry Association, 2001)

When outsourcing software applications through SaaS there will be fewer servers to maintain and thus less capital expenditure to be undertaken by a company (Erlanger, 2005), as the maintenance, support and upgrades of the application and related hardware infrastructure is outsourced to the SaaS vendor (Software & Information Industry Association, 2001). The SaaS vendor can normally perform this function in a much more cost effective manner, resulting in cost savings for the customer. As a result, the remaining IT budget of the SaaS customer can be distributed amongst the core business applications of the company, which contributes more directly to its bottom-line profits (Sysmans, 2006). IT departments of SaaS customers would therefore have the opportunity to focus on managing the services that software applications provide, rather than on implementing and maintaining these applications. By doing so, the IT department can contribute more directly to the primary business objective of the company (Carraro & Chong, 2006b).

Although many companies have avoided SaaS due to concerns such as security of data (Bleicher, 2006), integration with existing systems, customisability (Carraro & Chong, 2006b), the uncertainty surrounding long term costs (Fonseca, 2008), the need for the application to be available at the performance level of a traditional on-premise application and loss of control of applications, which are core to their business, to an outside party (Zucco, 2006). If implemented correctly, SaaS offers improved levels of efficiency, lower levels of risk and a lower total cost of ownership (Zucco, 2006).

For a service customer the most difficult part of determining whether to move to SaaS is the total cost of ownership ("TCO") calculation. This is due to the fact that the TCO does

not merely include the cost of new licences, but also the cost relating to integration, support, training and operations. (Gruman, 2007b)

2. Research problem and objective

As is evident from the above, there are some very compelling reasons for user companies to move toward the SaaS application delivery model. There are however numerous variables that should be considered by the potential SaaS customer, before this decision can be made.

These include, but are not limited to, the following:

- Data architectures (Chong, Carraro & Wolter, 2006);
- Connectivity (Software & Information Industry Association, 2000c);
- Customisability versus configurability (Dagum, 2006);
- Security and privacy of data (Bleicher, 2006);
- Availability of services and data (Zucco, 2006);
- Service levels (Wailgum, 2008);
- Data ownership (Zucco, 2006);
- Integration with existing systems (Carraro & Chong, 2006b); and
- Costing (Sysmans, 2006).

At present there are no clear and comprehensive guidelines or frameworks that identify all the relevant considerations to be taken into account by a potential SaaS customer in the decision to adopt SaaS. A need was thus identified for a holistic view of the considerations for the adoption of SaaS, as a viable alternative, to the traditional on-premise application platform.

The objective of this research is to provide a comprehensive framework of the main considerations to be taken into account by any company (service customer) considering the SaaS delivery model as an alternative to traditional on-premise software installations, to aid in this decision making process. In addition, this framework can also be used by the potential SaaS vendor in determining the viability of their SaaS offering, when compared to an equivalent, on-premise based software solution.

3. Research design and methodology

The study consisted of an applied research design focussing on an empirical study of existing data.

In order to compile the framework a detailed study of existing research on SaaS was undertaken to gather sufficient information regarding the considerations relating to the adoption of SaaS by a service customer. Firstly, the basic definition and characteristics behind SaaS was investigated, including the approaches to the delivery of SaaS in comparison to other software delivery models (chapter 2).

Following this, the drivers for adoption and limiting factors of SaaS was examined. Particular attention was paid to connectivity, customisation, availability, security, integration and scalability (chapter 3 and 4), as these issues represent the most important issues when considering a move to SaaS by a customer. For each of these components the cost implications for the potential SaaS customer were investigated (chapter 5) to enable the compilation of a Total Cost of Ownership ("TCO") calculation, which is presented in chapter six.

Chapter 2 – What is Software-as-a-Service?

1. Introduction

In this chapter the definition of SaaS and the various characteristics of SaaS (section 2), will be discussed. In addition section 3 of this chapter provides an overview of the current adoption of SaaS by service customers.

2. Defining SaaS

The main characteristics of SaaS can be defined as follows:

2.1 SaaS is an application delivery model whereby a commercial (not custom) application is hosted centrally by a SaaS service provider at their data centre (Software & Information Industry Association, 2000a);

2.2 The application is accessed across the Internet, a Local Area Network ("LAN"), an Intranet or via a Virtual Private Network ("VPN") (Software & Information Industry Association, 2001). SaaS applications are normally written as modern, Web-native applications that are accessed with a standard Web browser (Bleicher, 2006);

2.3 In the "pure" form of SaaS a single instance of an application is hosted by the SaaS vendor and made available to multiple customers, commonly known as multi-tenancy (Knorr, 2006);

2.4 As no software is installed locally, all upgrades and patches are done seamlessly from a central location (Carraro & Chong, 2006b);

2.5 Customers do not pay to own the application, but pay a subscription-based, usage-based, transaction-based, value-based or fixed fee (Software & Information Industry Association, 2001).

Each of these characteristics will be discussed in more detail in the section that follows.

2.1 Hosting of SaaS applications

Various applications can be hosted by a service provider under the SaaS software delivery model. According to Carraro & Chong (2006a), SaaS application offerings can be divided into two major categories:

- Consumer oriented services; and
- Line-of-business software ("LOB").

Consumer oriented services

A SaaS application can be as simple as web based messaging, such as Gmail, Hotmail and Yahoo mail. These applications meet all the basic characteristics of a SaaS application: a service provider hosts the application centrally and provides access to the data and application over the Internet, though a web browser. (Carraro & Chong, 2006a)

Consumer oriented services are offered to any member of the general public. Although these services are sold on a subscription basis in certain instances, these services are mostly offered at no charge and are funded by advertising revenue. These services are commonly referred to as "Web 2.0" applications. (Carraro & Chong, 2006a)

Line-of-business software

LOB software is normally aimed at companies (service customers) and includes applications to replace traditional business software normally hosted in companies' IT centres (Carraro & Chong, 2006a) such as applications for back office systems like enterprise resource planning and human resources, messaging, integration and CRM (Erlanger, 2005) applications. These LOB SaaS applications are normally provided on a subscription basis (Carraro & Chong, 2006a).

<u>Back office</u>

Back office applications that can be outsourced through SaaS include enterprise resource planning ("ERP"), purchasing, human resource ("HR") management and payroll applications. Traditionally ERP and related applications were thought of as incompatible with the philosophy of SaaS; however large software vendors such as Oracle and SAP are

providing these applications through SaaS to approximately 8% of their customer base. (Erlanger, 2005)

Messaging

Messaging has become an administration burden for many companies. A large portion of IT resources of companies are devoted to the management of e-mail, including managing mail boxes, archiving mail, filtering viruses and spam and adhering to regulatory requirements (Erlanger, 2005). These types of messaging services are suitable for outsourcing to a SaaS service provider.

Integration

More than half of the current SaaS subscribers use at least three different types of applications delivered via SaaS. As the adoption of SaaS increases, so will this number and as a result the integration between these applications will become increasingly complex, thus re-introducing complexities that SaaS was intended to eliminate in the first place. (Erlanger, 2005)

The solution may be to have an integration application that is also delivered via SaaS. Applications for integration are already available through Grand Central, which offers a business-to-business integration platform. (Erlanger, 2005)

Integration and related issues are discussed in detail as part of chapter 4.

Customer relationship management (“CRM”)

Currently, CRM forms the largest part of the SaaS arena (Gruman, 2007a). As this area of SaaS expands, many vendors are providing industry specific CRM solutions, which can help reduce the costs associated with customisation (Erlanger, 2005).

2.2 Approaches to delivering SaaS

Potential SaaS customers have to understand the various ways in which software vendors can approach the delivery of SaaS, to enable customers to make an informed decision as to the choice of the specific SaaS solution.

Certain vendors adapt their existing products to incorporate SaaS elements. Other vendors redesign their entire existing product lines for the SaaS market. Other vendors create specifically new SaaS enabled applications. In addition to vendors adopting products for SaaS, various new companies have been established over the past few years, which develop applications specifically for SaaS. (Software & Information Industry Association, 2000a)

As mentioned earlier various approaches to deliver SaaS exist. In this section physical connectivity and application delivery technologies (logical connectivity) utilised in the delivery of SaaS will be discussed. However, it should be noted that physical connectivity will only be discussed at a high level, as the definition of SaaS will inherently result in customers utilising existing internet connections, rather than installing expensive leased lines to SaaS vendors.

Physical connectivity

Connectivity is one of the most critical issues for potential SaaS customers to consider. Customers using SaaS should be able to connect to the central application from anywhere at anytime and using any type of device, from mobile devices to desktop computers, without sacrificing functionality. The speed of the network connecting the SaaS vendor to the customer is therefore of utmost importance. Without high-speed, always-on, 100% reliable physical connections to the centralised data storage and service provider, SaaS will not be able to operate effectively. Even though the speed of Internet connections have increased and the use of broadband connections are more common place, most users still use relatively slow Internet connections. (Software & Information Industry Association, 2000c)

It is therefore important for the potential SaaS customer to carefully decide on the physical connection medium to the SaaS vendor. The Software & Information Industry Association (2001) defines the following four possible mediums available to SaaS customers for physical connections to a SaaS vendor:

i) Internet;
ii) Local Area Network (“LAN”);
iii) Intranet; or
iv) Virtual Private Network (“VPN”).

The choice of physical connection will have a definite impact on the choice of the logical connection method. As a result these two aspects should be considered in conjunction with each other.

Application delivery technologies (logical connections)

Application delivery refers to the manner in which the SaaS customer will access the SaaS application over the physical connection medium. According to the Software & Information Industry Association (2001) there are four main manners in which SaaS applications can be delivered:

i) Server-based computing, also known as thin-client computing;
ii) Hosted client computing ("HCC");
iii) Web-based access; and
iv) Java applications.

Server-based computing (thin-client computing)

In a thin-client computing environment, users move from full-featured computers to lightweight machines which require less maintenance and fewer upgrades and are primarily used for display and input. Companies then provide computing services to their end users' thin clients from high-powered servers over a network connection. More effective utilization of computing hardware is achieved as server resources can be shared across many users. (Nieh, Yang & Novik, 2000)

The typical thin-client platform consists of a client application that is executed on a user's local desktop machine and a server application that executes on a remote system. The end user's machine can be a specialised hardware device designed to run the client application. Alternatively, a low-end personal computer can also be utilised for this purpose by the end-user. The remote server typically runs a standard server operating system, and the client and server communicate across a network connection between the desktop and server. The only information exchanged is client input data sent across the network to the server, and display updates which the server then returns to the user's machine. (Nieh*, et al.,* 2000)

The SaaS application can therefore be run on a centrally maintained server and the user interface can be provided to the user through a thin-client. The output of the user interface is accessed through a special client program or browser. (Software & Information Industry Association, 2001)

Hosted client computing ("HCC")

The application is downloaded into the memory of the user's desktop computer and information is streamed from a central server. When the user has completed the session, the application is removed from the user's machine. More efficient HCC applications only stream the components that are needed in a session to the user's machine. (Software & Information Industry Association, 2001)

Web-based access

These applications are deployed via HTML and maintained on a central server and viewed through a web-browser. With this method, the user interface logic is simplified and the application and user data is separated. (Software & Information Industry Association, 2001)

In the "pure" form of SaaS, the application is web-native, in other words it was designed to be accessed via a web-browser over the Internet (Bleicher, 2006). With this method of access, a standard web-browser can be used to connect to the remote application. Most desktop computers come pre-installed with a web-browser such as Internet Explorer or Mozilla Firefox. All the end-user has to do is enter the web-address of the SaaS application in the browser to access the remote application. Data and functionality can then be accessed through the standard user interface of the web-browser. The application is never physically downloaded to the user's machine, but is streamed from the remote server.

Java applications

Java is an extension to the traditional web-based application access method as described above. "Java allows applications to include a rich mix of interactive features and functions that simple Web– or HTML-based applications cannot offer." (Software & Information Industry Association, 2001)

Java applications are commonly referred to as applets. “An applet is a program written in the Java programming language that can be included in an HTML page, much in the same way an image is included in a page. When you use a Java technology-enabled browser to view a page that contains an applet, the applet's code is transferred to your system and executed by the browser's Java Virtual Machine (JVM).” (Sun Microsystems, 2008). This will mean that certain of the application’s functions are performed locally, thus reducing the amount of data that has to be sent back and forth across the network connection.

2.3 Multi-tenancy

One of the defining characteristics of SaaS is its ability to operate in a multi-tenancy manner

2.3.1 Definition

In the “pure” form of SaaS a single instance of an application is hosted by the SaaS vendor and made available to multiple customers, by sharing the physical servers and related hardware systems among the tenants, although the servers are logically separated for each tenant. This is commonly known as multi-tenancy.

According to Smoothspan (2007) multi-tenancy is “the ability to run multiple customers on a single software instance installed on multiple servers to increase resource utilization by allowing load balancing among tenants, and to reduce operational complexity and cost in managing the software to deliver the service. Tenants on a multi-tenant system can operate as though they have an instance of the software entirely to themselves which is completely secure and insulated from any impact by other tenants.”

To fully understand the above definition, one also needs to define the concepts of “users” and “tenants” as used in the context of multi-tenancy (Chong, *et al*., 2006):

- “Tenant” refers to the company using the SaaS application (service customer) to access its own data, which is logically separated from other tenants’ data.
- “User” refers to the individual end users of the tenant or company, who can gain access to the data of that tenant, through user accounts that are controlled by the tenant.

Multi-tenancy is contrasted with single tenant architectures where each tenant has access to a separate instance or installation of an application that is hosted on a separate server by the service provider. Each tenant or customer only has access to the server and application assigned to them. (Sysmans, 2006; Knorr, 2006)

The main driver behind multi-tenancy is the ability to achieve greater cost savings than through the consolidation of IT resources into one single operation (the SaaS vendor) alone (Carraro & Chong, 2006a). These costs savings are achievable due to the fact that the licensing costs of the underlying software (including the operating and database management system) as well as the operational costs of the application are spread across more than one tenant. These costs savings can then be passed on to the customers and the SaaS vendor has an opportunity to generate bottom-line profits. (Smoothspan, 2007)

2.3.2 Various approaches to multi-tenancy

In the multi-tenancy environment, additional development is required to ensure that each tenant's data is fully secure from other tenants' data and that the data and application are available to the particular SaaS customer when needed (Bleicher, 2006). Two broad categories of approaches can be identified (Chong*, et al.,* 2006; Carraro & Chong, 2006a):

- The isolated approach; and
- The shared approach.

It is important for the potential SaaS customer to understand the various approaches to multi-tenant data architecture, as this will directly impact on considerations such as:

- Data isolation and security;
- Customisation;
- Backup and recovery; and
- Cost

In the sections that follow the impact of the various approaches on each of these considerations will be highlighted.

Isolated approach

This approach is also referred to as the “separate tenant database” approach. This is the least complicated approach to isolate data. Each tenant’s data is stored in a separate database. The physical computer resources and application code is shared amongst the tenants on a server. Each tenant’s data is logically separated from the other tenants’ data through the use of metadata that links each database to the correct tenant. Database security ensures that tenants can not maliciously or accidently access other tenants’ data. (Chong, *et al.,* 2006)

This approach can be graphically represented as per figure 1 below.

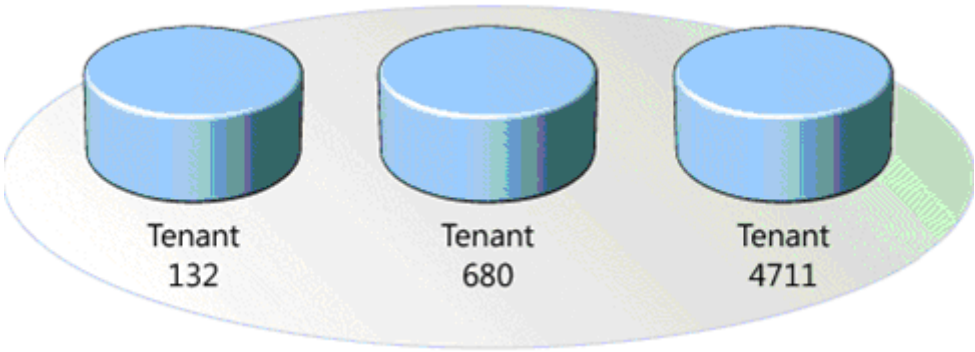

Figure 1 Separate tenant database (Source: Chong, *et al.,* 2006)

The major characteristics of this approach can be summarised as follows:

- The tenant is able to modify the database as far as the application’s program logic and user interface allow. The tenant is able to create new fields, queries, tables and relationships. (Carraro & Chong, 2006a)
- This approach provides the maximum security and data isolation that is required by customers with strict data isolation requirements (Carraro & Chong, 2006a). This model is therefore appropriate for customers who are willing to pay for the additional security that this model offers (Chong, *et al.,* 2006). These customers normally include banking institutions and medical record management companies (Carraro & Chong, 2006a).
- This approach also simplifies the process for data backups and restores (Chong*, et al.,* 2006), due to the fact that each customer’s (tenant’s) database is physically separated from databases of other customers.
- This approach leads to higher overall costs. As only a limited number of tenants can be hosted on a physical server, the hardware and maintenance costs per tenant are higher. (Chong*, et al.,* 2006)

Shared approach

The shared approach basically relates to the fact that customers' databases are not necessarily physically separated. The following three distinct techniques relating to the shared approach can be identified (Chong, *et al.,* 2006; Carraro & Chong, 2006a):

i) Shared database, separate schemas with a fixed extension set;

ii) Shared database, shared schema with a fixed extension set; and

iii) Shared database, shared schemas with custom extensions.

Shared database, separate schemas with a fixed extension set

With this approach all tenants share the same database and a single code base for the application. A schema consisting of a group of tables is set-up for each tenant. (Chong*, et al.,* 2006) This schema includes custom fields that are preset for each tenant. The tenant is then able to use and assign these fields based on their own requirements (Carraro & Chong, 2006a). For a graphical representation refer to figure 2 below.

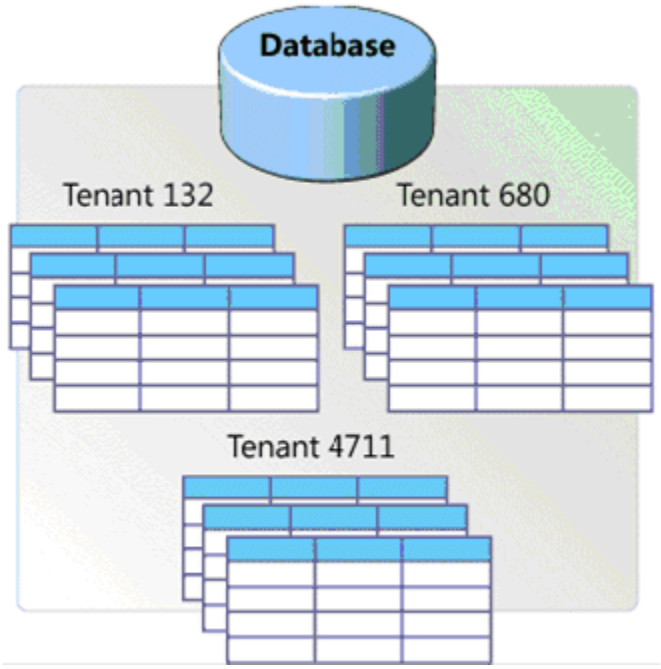

Figure 2 Shared database, separate schemas with a fixed extension set (Source: Chong, *et al*., 2006)

The major characteristics of this approach can be summarised as follows (Chong*, et al.,* 2006):

- This model provides only a moderate level of data isolation from other tenants' data as tenants share the same database.

- The data restoration process is more complicated under this approach than under the separate database approach. In the event of a hardware failure, the data of every tenant on the database server would have to be overwritten with the backup data, irrespective of data loss suffered. In effect, the database administrator would have to restore the data to a backup server and then import each of the tenant's database tables to the production server. This can be a complicated and expensive procedure.
- The customisability of the fixed extension set is limited to the number of custom fields provided to the tenants. Determining the number of fields to include in the extension set is difficult, as it would be impossible to accurately provide for each customer's needs. If too many fields are included, the database may contain unused space and effectively be costly to maintain. If too few fields are provided the customers may not be able to customise the database to their specific needs.
- As this approach can support a larger amount of tenants per database server, it results in lower overall costs to the customers. Therefore, this model would be appropriate for cost conscious customers who can accept that their data will be hosted in the same database as other customers, but who still require a moderate level of data isolation and security.

Shared database, shared schema with a fixed extension set

This approach involves hosting all tenants within the same database, *sharing* a set of database tables. Effectively, each table in the database can include records from more than one tenant, not stored in any specific order. An additional field in the table, for example a "tenant ID" field, would associate each record with the correct tenant (Chong, *et al.,* 2006). For a graphical representation refer to figure 3.

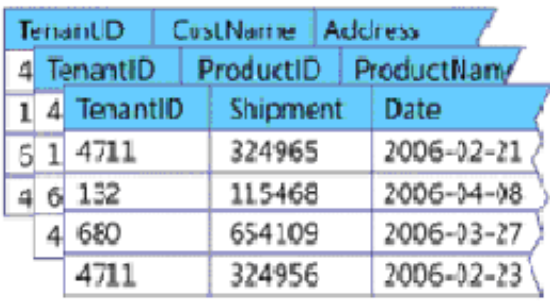

Figure 3 Shared database, shared schema with a fixed extension set (Source: Chong, *et al*., 2006)

The major characteristics of this approach can be summarised as follows (Chong*, et al.,* 2006):

- Under this approach, the tenants will share a schema and will therefore not be able to customise the extension set of the database.
- Due to the inherent complexity of this model, the SaaS vendor would have to incur additional development costs to ensure that tenants' data is appropriately secured from unintentional or malicious access. Data isolation is thus not as robust as with the other approaches.
- The shared schema approach introduces an additional complexity during data restoration as the backup data would first have to be restored to a backup server, and then individual rows would have to be deleted from the production database. Once this process is completed, the affected rows would have to be re-inserted from the backup server into the production database. This can cause performance degradation if a large number of rows are affected and can also lead to additional costs.
- As this approach can serve the largest number of tenants per database server, it has the lowest overall hardware and maintenance costs. This approach is therefore appropriate where low cost is more important to the SaaS customer than data isolation. It is important that the SaaS application is able to support a very large amount of users, on a limited number of servers.

Shared database, shared schemas with custom extensions

This approach is essentially the same as the shared schema approach discussed above. The main difference is that with this model, the extension set is not fixed, thus allowing tenants to extend the data schema without limits, whilst still retaining the cost benefits of a shared database. Any changes made to the data model would be stored as metadata in a separate database table. (Carraro & Chong, 2006a)

With this approach the same disadvantages are experienced as with the shared schema approach mentioned above. Normal database functions such as searching, indexing, querying and updating records are also complicated due to the complexity of this approach. This is mainly due to the fact that any change made by one of the tenants would require the business logic of the application and the presentation logic of the application to be amended. (Carraro & Chong, 2006a)

This model would be appropriate for customers who require a large degree of customisability with regards to their data models, but who do not require data isolation (Carraro & Chong, 2006a).

2.3.3 Choosing the most appropriate approach to multi-tenancy

The choice of approach for multi-tenancy should be taken by the SaaS service provider, as this will directly impact on the delivery model for SaaS. The choice of data architecture will also determine to an extent, the type of customer that will be serviced.

According to Chong, *et al* (2006) the following considerations should be taken into account when a SaaS vendor decides which multi-tenancy approach would be most appropriate:

- Economic considerations;
- Security considerations;
- Tenant considerations;
- Regulatory considerations; and
- Skill set considerations.

The effect of each of these considerations on the approach is summarised in table 1 in section 2.3.4.

Economic considerations

As discussed under the approaches to multi-tenancy above, due to inherent complexities, the shared approach requires extensive development up-front, resulting in increased costs. However, as a larger base of tenants can be hosted per server, the ongoing operational costs are lower than that of the isolated approach. (Chong*, et al.,* 2006)

This is illustrated by figure 4 below.

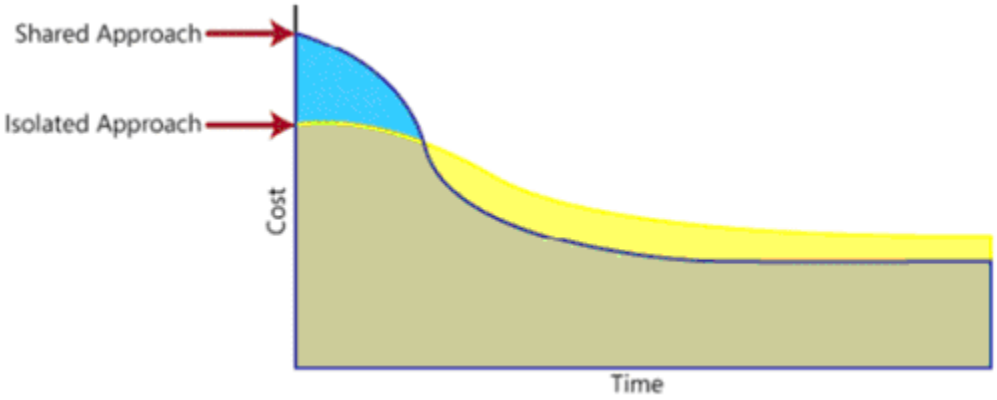

Figure 4 Ongoing operational costs per method (Source: Chong, *et al*., 2006)

Security considerations

As the SaaS database will house sensitive tenant data, customers will expect high levels of security. These security levels and related guarantees will be contractually agreed in the form of a Service Level Agreement ("SLA") with the service provider. (Chong, *et al.,* 2006; Carraro & Chong, 2006b)

Physical isolation (physically separating the database servers) of data is considered by many to be the only approach to provide sufficient data security. However, a shared approach can also provide a sufficient level of security, but does require significantly more development to implement. (Chong*, et al.,* 2006)

Tenant considerations

Several items should be considered within this category (Chong, *et al.,* 2006):

- The number of expected tenants
    - The more tenants are expected to be serviced as part of the strategy, the more appropriate a shared approach would be. This will ensure that costs are limited.
- The number of users per tenant
    - The more end-users are expected, the more an isolated approach would be appropriate to meet user requirements.
- The amount of data to be stored
    - A more isolated approach would ensure better service where large amounts of data are to be stored for tenants.

- Per tenant value added services
    - If value added services, such as per tenant backup and restore facilities are offered, a more isolated approach would be appropriate.

Regulatory considerations

In certain jurisdictions, regulatory law may prescribe the security and record storage requirements to companies (Chong*, et al.,* 2006). This would have an impact on the choice between the isolated and shared approach. Refer to chapter 3, section 3(i) for more information.

Skill set considerations

As the shared approach involves a considerable development effort, an isolated approach may allow the staff of the SaaS vendor to use existing knowledge of software development (Chong*, et al.,* 2006), resulting in a more efficient deployment of the SaaS offering.

2.3.4 Summary of considerations per approach

There are various considerations that would influence the choice for the most appropriate approach for multi-tenancy. These are summarised in Table 1 below.

Table 1 Summary of considerations per approach

Description	**Isolated approach**	**Shared approach**		
	Separate tenant databases	**Shared database, separate schemas with a fixed extension set**	**Shared database, shared schema with a fixed extension set**	**Shared database, shared schemas with custom extensions**
Simple to implement?	Yes	Yes	No	No
Lower costs?	No	Yes	Yes	Yes
Simple to restore data?	Yes	No	No	No
Level of data isolation?	High	Moderate	Low	Low
Amount of data that can be stored?	High	Moderate	Low	Low
Meets regulatory requirements for strict data isolation?	Yes	No	No	No
Level of customisability?	High	Low	Low	High

Description	Isolated approach	Shared approach		
	Separate tenant databases	**Shared database, separate schemas with a fixed extension set**	**Shared database, shared schema with a fixed extension set**	**Shared database, shared schemas with custom extensions**
Performance?	High	Moderate	Low	Low
End users per tenant?	High	High	Low	Low

This table was compiled from the information in this chapter and serves to show that it is important for the SaaS vendor as well as the potential SaaS customer to consider all variables and not only the cost implications of a specific approach to multi-tenancy. The specific approach selected should be based on consideration of all the variables involved as well as the specific needs of each prospective customer.

2.4 Customisation and configuration

SaaS applications normally do not allow for customisation of the application through source code changes. As a result there are less customisation options than with a traditional on-premise application (Dagum, 2006). In addition, one of the features of multi-tenancy is that any change to the source code is immediately available to all the other users of the SaaS application (one instance, many tenants) (Carraro & Chong, 2006a).

This limited scope for configuring and customisation results in lower development costs for the SaaS vendor and faster implementation times for the SaaS customer. However, this limited scope for customisation may not meet the critical requirements of companies in terms of automation and business intelligence (Dagum, 2006). Customers would therefore be more likely to *configure* an application, through the use of metadata, rather than customise the application to suit their needs. It is therefore important for the SaaS application to provide a configuration interface that would allow users to be able to perform these functions themselves. (Carraro & Chong, 2006a)

The National Information Standards Organization (2004) states that metadata is “structured information that describes, explains, locates, or otherwise makes it easier to retrieve, use, or manage an information resource.” Metadata is often referred to as “data about data or information about information.”

According to Carraro & Chong (2006a), SaaS customers can typically apply configuration changes in four broad areas:

i) User interface and branding: Users can modify the user interface to reflect their corporate branding. Normally customers can change aspects such as graphics, colours, fonts, etc.

ii) Workflow and business rules: Typically, customers would want to align the application's workflow with current business processes to ensure adherence to company policies.

iii) Extensions to the data model: A data model that can be extended enables customers to use an application in the manner which they prefer; instead of the application forcing the customer to work its way. This is particularly relevant as many data-driven SaaS applications' data models do not suit every single customer.

iv) Access control: Each tenant or customer is responsible to ensure that user accounts are created for each individual user that will be utilising the SaaS application, and that these access rights and restrictions conform to the company policies. As a result it is important that these rights and restrictions be configurable by the customer.

2.5 Subscription models

Various SaaS subscription models are available. These subscription models can be divided into the following four categories as explained by the Software & Information Industry Association (2001):

- Subscription based model
    - With this model a monthly payment is made based on the software actually used. It normally includes a commitment for the number of actual end-users of the application. These subscriptions are normally billed on a per-seat or named user basis.
- Usage based model
    - Payment is determined based on the actual usage of the application and is related to peak levels of usage. This method may also be based on the number of servers that run the hosted application or the number of concurrent users.

- Transaction based model
    - Customers are charged for each transaction that is processed via the hosted application (Carraro & Chong, 2006b).
- Value based model
    - Payment to the SaaS vendor is reliant on the achievement of an agreed business goal for the customer.
- Fixed fee model
    - Monthly payments are set up-front based on the number of users, support levels required and modules of the application that will be used (Carraro & Chong, 2006b).

3. SaaS adoption

Various software applications are suitable for outsourcing via SaaS. Figure 5 presents the percentage of current SaaS applications per application type. As can be seen from the figure, 18% of current SaaS applications are Customer Relationship Management ("CRM") applications. This is closely followed by Human Resources (15%) and Procurement applications (12%). In terms of customers which have adopted SaaS, technology, financial services and utility companies are on the fore-front.

It is expected that the SaaS market will grow by 22.1% to 2011 (Lauchlan, 2007), when the total SaaS market is predicted to reach $19.3 billion (Lawson, 2007). It is also expected that 25% of business software will be delivered via SaaS in 2011 (Finch, 2008).

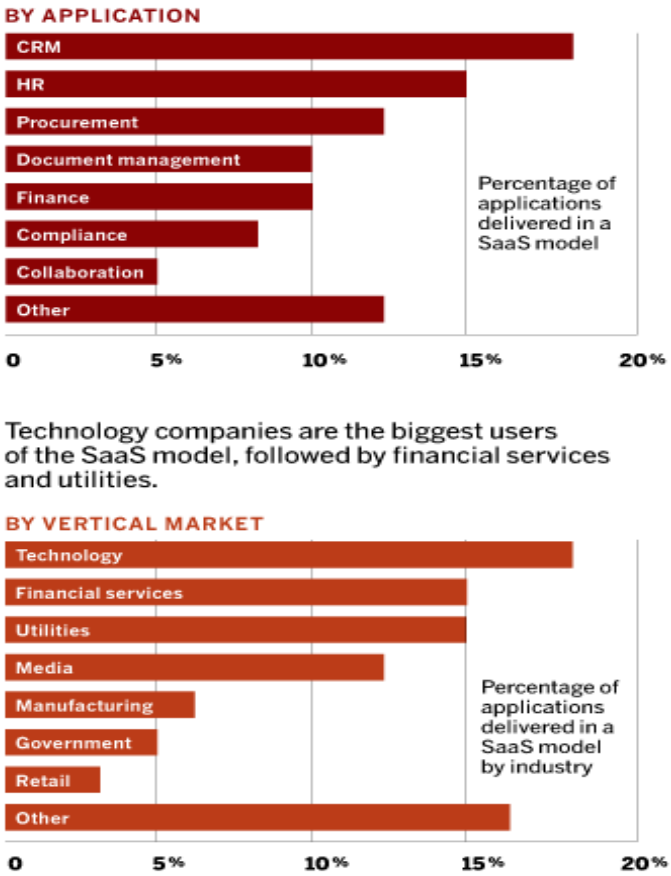

Figure 5 Current usage of SaaS applications (Source: Gruman, 2007a)

4. Summary

This chapter focussed on explaining and discussing the definition of SaaS, the characteristics of SaaS as well as the current levels of SaaS adoption and predictions for its future. The drivers for the adoption of SaaS as well as the limiting factors and challenges faced by SaaS customers when implementing SaaS as a viable alternative to the traditional software delivery model, will be examined in the following chapters.

Chapter 3 – Advantages and limitations of SaaS

1. Introduction

To be able to make an informed decision whether to adopt SaaS or not, one should be familiar with both the advantages and disadvantages of SaaS. As a result of this, the various reasons why customers would want to move away from the traditional shrink-wrapped, on-premise based applications to the SaaS software delivery model will be explored in this chapter. The limitations, or disadvantages, of SaaS as a software delivery solution will also be discussed.

2. Drivers for adoption

There are numerous advantages and drivers for the adoption of SaaS by a service customer as an alternative to the traditional on-premise based application delivery model. The main advantages and drivers for adoption of SaaS can be summarised as follows:

i) Shorter implementation time (Bleicher, 2006);
ii) Vendor alignment with customer (Sysmans, 2006);
iii) Changing the role of IT to a value added service (Sysmans, 2006);
iv) Cost reduction (Carraro & Chong, 2006b);
v) Access to application functionality previously reserved for larger companies (Software & Information Industry Association, 2000b);
vi) Easier compliance with regulations (Zucco, 2006);
vii) Reduced interruptions due to upgrades (Software & Information Industry Association, 2000b); and
viii) Scalability of SaaS applications (Sysmans, 2006).

Each of these items are discussed in the remainder of this chapter.

i) Shorter implementation time

Traditionally, software deployment within companies can take years, consume a large quantity of available resources and may not provide the desired result (Zucco, 2006). The time between selecting and implementing an application is drastically reduced under the SaaS model (Bleicher, 2006). If custom integration is not required, SaaS applications can be rolled out in a very short time period, delivering a very short time-to-value period for a

large IT investment. Some SaaS vendors even offer a free trial period to their potential customers to test their application (Carraro & Chong, 2006b). In addition, once a product is ready for distribution, customers would be able to access it through an existing Internet connection, doing away with a complex and time consuming implementation process (Software & Information Industry Association, 2000b).

ii) Vendor alignment with customer

Due to the low cost of switching between SaaS applications, as opposed to traditional software, SaaS vendors are motivated to align themselves fully with their customers and demonstrate the value that they can bring to their customers (Bleicher, 2006). This is mainly due to the fact that SaaS applications are priced on a subscription based model, which is linked to a contract term. As a result customers have a greater degree of control over SaaS vendors, holding the vendors to monthly service level agreements. If service levels are not adhered to, the customer can simply cancel their monthly payments to the SaaS vendor, enforce the terms of the SLA or move to a different vendor. The SaaS vendor therefore has a financial incentive to provide adequate support and ongoing maintenance to the customer. (Sysmans, 2006)

iii) Changing the role of IT to a value added service

In a SaaS environment the day-to-day operation of the SaaS application is completely administered by the SaaS vendor. This will include installing patches and upgrades, monitoring performance, ensuring high-availability and performing daily backups of data. By re-assigning these activities to the SaaS vendor, IT staff can move from being reactive and operations focussed, to supporting the business in achieving its goals and objectives. This enables the SaaS customer to focus on configuring the software to work with their own business processes (Bleicher, 2006). The SaaS vendor will also handle the support and training needs of the customer's end-users. This gives the IT department of the SaaS customer an opportunity to be a value-producing part of the business, rather than merely a cost centre. (Carraro & Chong, 2006b)

iv) Cost reduction

Traditional software applications, especially large ERP and CRM applications which are deployed across a large company, can cost hundreds of thousands of dollars in licensing fees, IT personnel, and consultants and will also require expensive customisation and integration with existing systems and data. SaaS applications do not require the

deployment of infrastructure (hardware) at the customer's premises, which dramatically reduces the initial costs associated with the deployment of a SaaS application. (Carraro & Chong, 2006b) SaaS therefore requires a significantly lower initial investment when compared to traditional software applications as all hardware and personnel costs are carried by the SaaS vendor (Wainewright, 2008).

In addition, with a SaaS application there is no up-front license fee payable (Gruman, 2007a). Subscription fees are normally billed on a per-month, per-use or per-transaction basis (refer to chapter 2) and all overheads in a SaaS model would be carried by the SaaS vendor as part of the service provided. This enables the customer to produce more reliable expense budgets, as the costs are recurring. (Software & Information Industry Association, 2000b)

v) Access to application functionality previously reserved for larger companies

SaaS enables smaller companies to gain access to the functionality of applications that were previously too expensive to implement and too complex to maintain. Smaller companies can now access, through SaaS, the tools normally reserved for larger enterprises, to run their companies more efficiently. (Software & Information Industry Association, 2000b)

vi) Easier compliance with regulations

The reporting requirements associated with regulatory compliance, such as Sarbanes-Oxley, can result in a significant administrative burden for companies. By using SaaS, a company can be ensured that all their locations are using the same version of the application and, therefore, all their reporting will be in a consistent format. The administrative burden of ensuring that all transactions are logged for a compliance audit is also reduced as the responsibility for the application resides with the SaaS vendor. (Zucco, 2006)

vii) Reduced interruptions due to upgrades

As the application is hosted centrally, the SaaS vendor can seamlessly deploy upgrades and patches in real time without any interruption due to a time consuming upgrade process (Software & Information Industry Association, 2000b). New functions and innovations can be streamed to all customers immediately, ensuring that the user interface remains consistent (Knorr, 2006).

viii) Scalability of SaaS applications

A SaaS application can grow with the business of a company. As there is no hardware or other infrastructure to acquire, expanding a SaaS application is as simple as increasing the number of users per the agreement with the SaaS vendor. (Sysmans, 2006)

3. Limiting factors and other considerations

Even though there are many compelling reasons to move to SaaS, there are also certain limiting factors, or disadvantages, and challenges facing customers who adopt SaaS. These include:

i) Increased security risks(Bleicher, 2006);
ii) Political considerations (Carraro & Chong, 2006b);
iii) Technical considerations (Carraro & Chong, 2006b);
iv) Financial considerations (Software & Information Industry Association, 2000c);
v) Availability (Tarzey, 2008);
vi) Integration with existing systems (Carraro & Chong, 2006b);
vii) Unfamiliarity with SaaS (Software & Information Industry Association, 2000c);
viii) Actual deployment time (Sieper, 2007); and
ix) Outsourcing core business applications (Gruman, 2007a).

These items will be discussed briefly in the following section.

i) Increased security risks

With SaaS, critical business information may be stored off-site in a server facility that can not be seen or inspected (Bleicher, 2006). Storing the data off-site introduces the risk that data may be lost or accessed without approval (Carraro & Chong, 2006b). Large customers may be reluctant to trust a service provider with their data, without some assurance surrounding security practices (Bleicher, 2006).

In certain parts of the world, regulatory requirements place strict record keeping and data security requirements on companies. Many SaaS providers would at present be unable to adhere to these record keeping and internal data security standards. This is of particular concern to institutions such as banks, which would need to be satisfied that information

processing outside of their firewalls occurs in a well controlled environment, based on best practices as well as regulatory standards. (Carraro & Chong, 2006b)

ii) Political considerations

There may be resistance from within companies to outsource their applications to a third party, as the impression may be created that the IT department is losing control of the application. As a result, the decision to move to SaaS may be based on issues not relating to IT. (Carraro & Chong, 2006b)

iii) Technical considerations

Certain applications may be too complex to convert to SaaS and will also require specialised support, which the SaaS vendor may not be able to provide (Carraro & Chong, 2006b).

The type and amount of data that the application will have to transmit across the network is a very important consideration. The typical bandwidth of a local area network with a gigabit Ethernet link is multiple times higher than any Internet connection can provide at present. A data transfer that would have taken a couple of minutes between servers in a data centre might take hours to complete in the case of a SaaS application geographically separated from the customer. (Carraro & Chong, 2006b)

iv) Financial considerations

The initial cost of implementing a SaaS application may be significantly less than a traditional on-premise application; however the long-term costs involved are much less certain. Factors that would need to be considered include: the effort needed to fully integrate the application with existing systems, configuration required and the investment in existing on-premise applications. (Carraro & Chong, 2006b)

Many large companies which have invested substantial amounts in existing on-premise software will be hesitant to replace these systems completely and would therefore only use SaaS as a means to extend capabilities rather than replace them. (Software & Information Industry Association, 2000c)

v) Availability

Some valid concerns surrounding the ability of SaaS vendors to provide an “always-on” service to its customers exist. Any break in the connection or hardware failure at the SaaS vendor would mean that the application would be un-available for use. An application that is not available to a company, as and when needed, will lead to loss of revenue for that company. (Tarzey, 2008)

In cases where an Internet connection is temporarily unavailable, a company should be able to continue using the application. This implies that more than just the web browser would be required on the premises of the SaaS customer. For example, it should be possible to continue working on an application “off-line” and then synchronise any changes to SaaS application servers. A SaaS software delivery model may therefore not be practical at all times. (Tarzey, 2008)

vi) Integration with existing systems

The prospective SaaS application would need to be integrated fully with a possibly diverse set of existing applications (Carraro & Chong, 2006b). This will lead to additional time and costs associated with the deployment of the SaaS application. Integration is discussed in more detail as part of chapter four.

vii) Unfamiliarity with SaaS

Many companies are not aware of the various software application solutions available in the form of SaaS, as a viable alternative to traditional software solutions (Software & Information Industry Association, 2000c).

viii) Actual deployment time

Many SaaS vendors will claim that it is possible to have their SaaS application completely up and running within days. However, this does not take into account the time needed to fully configure and integrate the application, as to be useful to the customer. (Sieper, 2007) As a result the actual deployment time may be longer than quoted by the SaaS vendor.

ix) Outsourcing core business applications

It will be difficult to outsource an application that is core to the business of a company or which differentiates that company from its competitors. With SaaS, companies will have to use software that was designed for hundreds of other companies. As a result companies

will find it difficult to outsource ERP, business intelligence and manufacturing systems, as these applications are normally highly customised. In other situations, functions provided by software applications are so critical to the operation of a business that it has to be owned to ensure high levels of availability and security. (Gruman, 2007a)

4. Summary

As can be seen from the previous two sections, there are numerous valid reasons to move to SaaS, but substantial evidence opposing this view as well. It is therefore critical for the potential SaaS customer to fully understand both the pros and the cons of adopting SaaS, to be able to make an informed decision.

In the next chapter the major considerations that will have an impact on the operations and sustainability of a company will be discussed in detail. It is important for the potential SaaS customer to be aware of these issues as this will have a direct impact on their business and processes once they decide to adopt SaaS as a delivery model.

Chapter 4 – Major considerations for potential SaaS customers

1. Introduction

There are certain issues that represent significant concerns and hurdles that need to be overcome for SaaS to be a successful replacement for traditional on-premise software. As a result it is important for the potential SaaS customer to understand the impact of these issues on their choice to move to SaaS or not. These major considerations as identified from various sources will be discussed in this chapter, including:

- Availability of services and data;
- Security and privacy of data;
- Integration with existing systems;
- Scalability; and
- Service level agreements.

2. Availability of service and data

2.1 Introduction

With e-commerce it is possible to conduct business 24 hours a day and seven days a week. As a result, companies are becoming increasingly dependent on IT systems (IBM, 1998). As an outside entity is running the software application in a SaaS model, customers are dependent on the SaaS vendor to provide a highly available system. The fear therefore remains that the customer will not receive the uptime levels that they require. (Gruman, 2007a)

Availability refers to the fact that a system is on-line and can be accessed by users (IBM, 1998). A system is unavailable if a user cannot access the system due to hardware, operating system or application program failure. This is generally referred to as downtime (PC Magazine, 2008). Downtime can be caused by a variety of factors, from planned maintenance to a hardware failure caused by a disaster (IBM, 1998).

Guaranteed high levels of availability is therefore one of the most important considerations for the potential SaaS customer. Unscheduled downtime for any single server or data centre could lead to significant data, productivity and monetary losses for the entire

customer base. (Carraro & Chong, 2006a) Any downtime in IT systems can lead to significant losses for customers (IBM, 1998).

The potential SaaS customer should ensure that the SaaS vendor has a solution in place to minimise downtime and reduce the amount of time needed to recover from a break in availability. This solution should focus on preventing and avoiding possible problems that could lead to system downtime. It should also focus on limiting the damage that can be caused by unscheduled downtime. This approach requires the correct mix of hardware, software and services to be effective. (IBM, 1998)

From the above it is clear that availability is and should be of major concern for the potential SaaS customer when considering a move to SaaS. It is therefore not uncommon that levels of availability are specified in SLA's with SaaS vendors. SLA's will be discussed in section 6 of this chapter.

3. Database security and privacy of data

Data may arguably be the most important asset of a company, as it could include information about products, processes, employees, customers, financial results, suppliers, trade secrets and much more. Data is core to the fundamental principles of SaaS. Therefore if companies want to gain the benefits of using a SaaS application, they must be willing to surrender some of the control over their data and in doing so, trust the SaaS vendor to sufficiently secure their data. (Chong, *et al.,* 2006)

As a SaaS application is hosted from a central server, companies' data would be stored off-site and outside of their firewalls (Finch, 2008). This represents a significant security and privacy hurdle for many companies considering the move to SaaS. These companies are concerned that their data could be accessed inappropriately, changed or that sensitive customer data may be disclosed (Software & Information Industry Association, 2000c). In particular, companies are concerned about phishing, social engineering, denial of service attacks, and theft of sensitive information by employees of the SaaS vendor (Finch, 2008). As a result, many companies are not willing to give up control of their data and would rather invest in more expensive on-premise solutions (Software & Information Industry Association, 2000c).

In response to these concerns the SaaS vendors have to create “a SaaS data architecture that is both robust and secure enough to satisfy tenants or clients who are concerned about surrendering control of vital business data to a third party, while also being efficient and cost-effective to administer and maintain” (Chong, *et al.,* 2006). This data architecture has to consist of a comprehensive, multi layered security structure, which is constantly updated to protect against new threats. Furthermore, the SaaS vendor should continually communicate the security procedures in place to the customer. (Software & Information Industry Association, 2000c)

SaaS customers should ensure that the data security procedures similar to those in place for their own laptops, personal digital assistants and desktops are also applied by the SaaS vendor (Finch, 2008). Data protection guarantees and compliance with industry best practices by the SaaS vendor can assist in this regard. SaaS vendors should provide details about security procedures, disaster recovery plans and how data is secured, to each customer. (Wailgum, 2008)

Chong, *et al* (2006) states that “a secure SaaS application is one that provides defence in depth, using multiple defence levels that complement one another to provide data protection in different ways, under different circumstances, against both internal and external threats.” Building security into a SaaS application is one of the most important tasks of the SaaS architect, as a SaaS application’s data can include extremely sensitive information relating to a company and its customers. (Chong, *et al.,* 2006)

Due to the fact that the storage of business critical data takes place outside of the SaaS customer’s walls (physical and logical), this section will focus on the various security issues relating to data storage, as it pertains to SaaS.

Data storage and security will mainly be the responsibility of the SaaS vendor in a SaaS application model. It is however important for the potential SaaS customer to understand the security approach that the SaaS vendor will follow, as this directly impacts on the security and privacy of the customer’s data, due to the fact that the data will be stored on a database that is not under the direct control of the customer. Customers should also be satisfied that the security approach applied by the vendor is sufficient to meet their security standards and any related regulatory requirements. (Carraro & Chong, 2006b) Typically,

security requirements are included in the service level agreement with the SaaS vendor (Finch, 2008).

The security issue that specifically impacts SaaS, as discussed in this section, is database security (Chong*, et al.,* 2006) as this is one area that would not be under the direct control of the SaaS customer.

Chong, *et al* (2006) identify three patterns to provide the sufficient levels of security in a SaaS database:

A. Permissions – Using access control lists ("ACL") to determine who can access and use data, through the concepts of authentication and authorisation.
B. Filtering – Using an intermediary layer between the database and the tenant, to ensure that each tenant can only see their own data.
C. Encryption – Encrypting data in the database, so that it can not be used even if it is accessed by an unauthorised person.

A – Permissions

As described above, permissions are set up, using ACL's, to determine who can access and use data in a SaaS application. Chong, *et al* (2006) and Carraro & Chong (2006a) identify the following basic principles used in this method:

i) Authentication;
ii) Authorisation;
iii) Trusted database connections; and
iv) Secure database tables.

i. Authentication

With a SaaS model, each tenant is responsible to create and maintain user profiles for their individual users. This is known as delegated administration. Even though the tenant or customer creates the user accounts, the SaaS vendor still has to authenticate the user accounts (Carraro & Chong, 2006a). Authentication in a SaaS model normally takes the form of one of the following two methods (Carraro & Chong, 2006a):

a. Centralised authentication; or
b. Decentralised authentication.

However, in certain instances it may be more practical for a SaaS vendor to implement a hybrid approach, using the centralised authentication system for smaller customers and the decentralised authentication system for larger customers requiring single sign-on ("SSO") capabilities. (Carraro & Chong, 2006a)

a) Centralised authentication

In a centralised authentication system, the service provider creates a central user account database. Each tenant's administrator is granted access to this database to create, change and delete user accounts for their company. When a user logs on to the SaaS application, they have to provide their username and password, which is then authenticated against the central user directory. After successful authentication, the user is granted access to the application. (Carraro & Chong, 2006a)

This method does not require any additional infrastructure changes on the part of the tenant, as the entire authentication infrastructure is provided by the SaaS vendor. The disadvantage is that the implementation of SSO is considerably more difficult. (Carraro & Chong, 2006a)

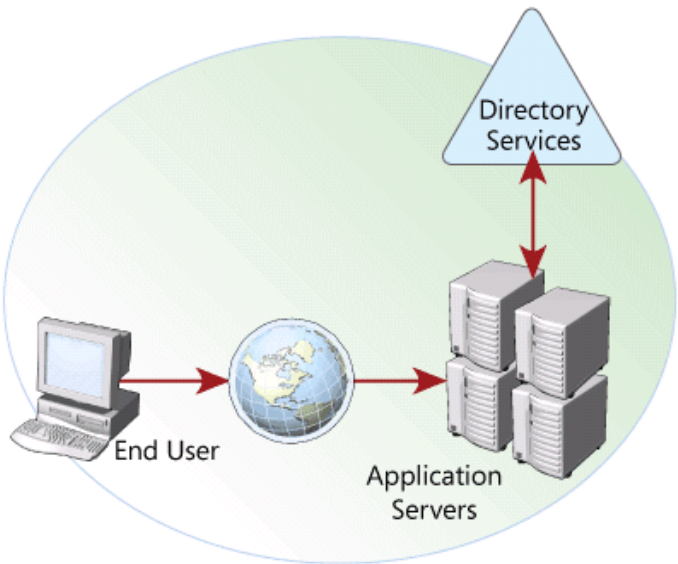

Figure 6 Example of a centralised authentication system (Source: Carraro & Chong, 2006a)

b) Decentralised authentication

With this system, each tenant deploys a federation service, which interfaces directly with the tenant's own directory service. The federation service will authenticate a user locally

when they attempt to access the application. Upon successful authentication, the local federation service will issue a security token, which the SaaS vendor's authentication system will accept to allow the user to access the application. (Carraro & Chong, 2006a)

This approach requires the customer to implement an additional federation system on their premises and thus incur additional costs. However, this approach is ideal where SSO is essential, as authentication is completed in the background. (Carraro & Chong, 2006a)

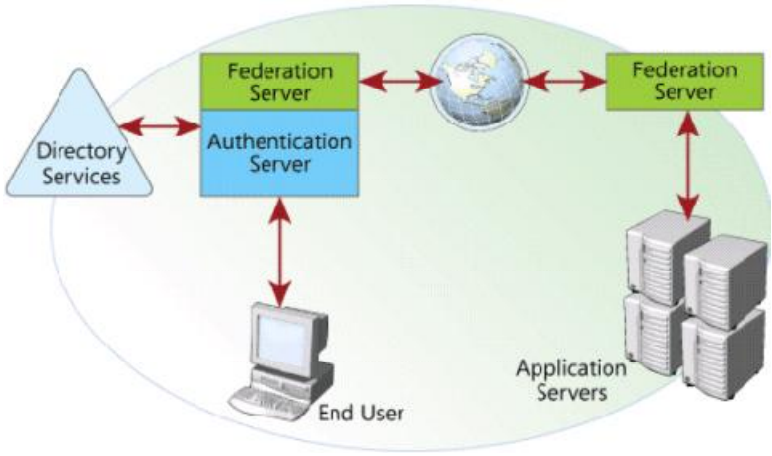

Figure 7 Example of a decentralised authentication system (Source: Carraro & Chong, 2006a)

ii. Authorisation

Once a user is authenticated, they require permission to use the application. This is typically known as authorisation to use the application and is determined by business roles and rules related to specific job functions in a company. These job functions are normally related to specific roles that are created within the application. Each role can include one or more permissions, which allow the user to perform certain tasks within the application. These permissions are mapped to specific business functions. Permissions may also relate to the management of the application itself. With a SaaS application tenants should be able to customise roles and permissions to suit their specific business needs. (Carraro & Chong, 2006a)

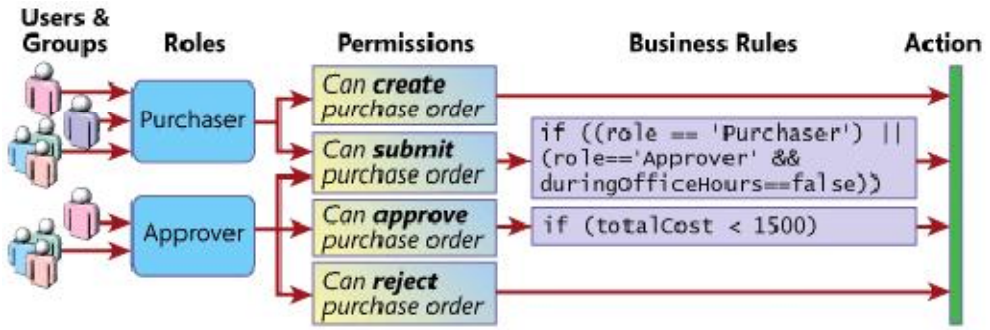

Figure 8 Illustration of roles and permissions (Source: Carraro & Chong, 2006a)

iii. Trusted database connections

Traditionally, two methods for securing data stored in a database are used: impersonation and trusted subsystem accounts. However, with the advent of SaaS a hybrid method may be more efficient and effective for the purposes of securing a database. (Chong, *et al.,* 2006)

a) Impersonation

This method requires each user to be set up on the database level to allow access to tables, queries, views, stored procedures and other objects. Whenever an end-user performs an action requiring a call to the database, the application will present itself to the database as the user, thus “impersonating” the user to be able to access the required object. (Chong, *et al.,* 2006)

This method requires a large amount of administration to ensure all users have the correct access set up on the database level. (Chong, *et al.,* 2006)

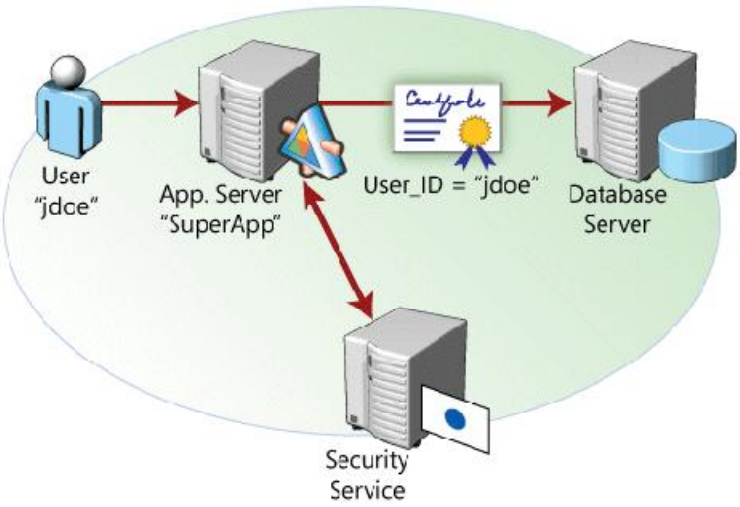

Figure 9 Illustration of Impersonation (Source: Chong, *et al.,* 2006)

b) Trusted subsystem access method

With this method the application will always connect to the database using its own process identity and not the identity of the actual end-user. The database server will only grant access to the database objects that the application is allowed to read or manipulate. Additional security has to be set-up at the application level to ensure users do not access objects which they are not allowed to. (Chong, *et al.,* 2006)

This method eliminates the need for each individual end-user to be set up at a database level and as a result requires much less administration. However, the ability to secure database objects for individual users is lost. (Chong, *et al.,* 2006)

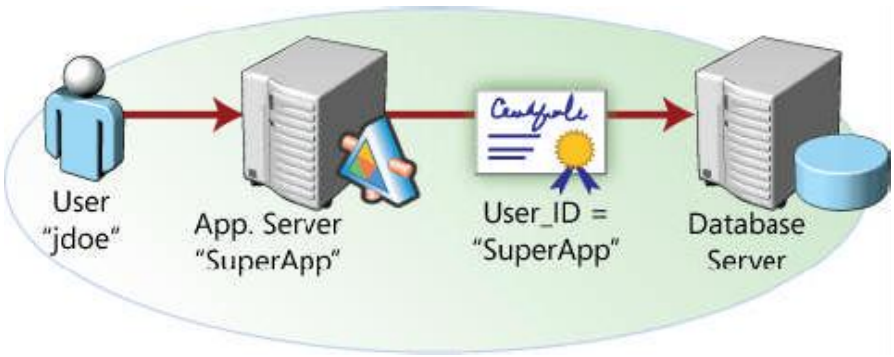

Figure 10 Illustration of the trusted database connection access method (Source: Chong, *et al.,* 2006)

c) Hybrid method: Impersonation and trusted subsystem

The hybrid method combines aspects from the impersonation and trusted subsystem methods. With this approach a database account is created for each tenant. ACL's are created to manage the access rights to database objects for these tenants. When an end-user performs an action requiring a call to the database, the application presents itself to the database as the *tenant* and not the individual user. The database therefore does not differentiate between different end-users within a tenant. Additional security has to be set up on an application level to ensure individual users within the tenant do not access objects which they are not authorised to. (Chong, *et al.,* 2006)

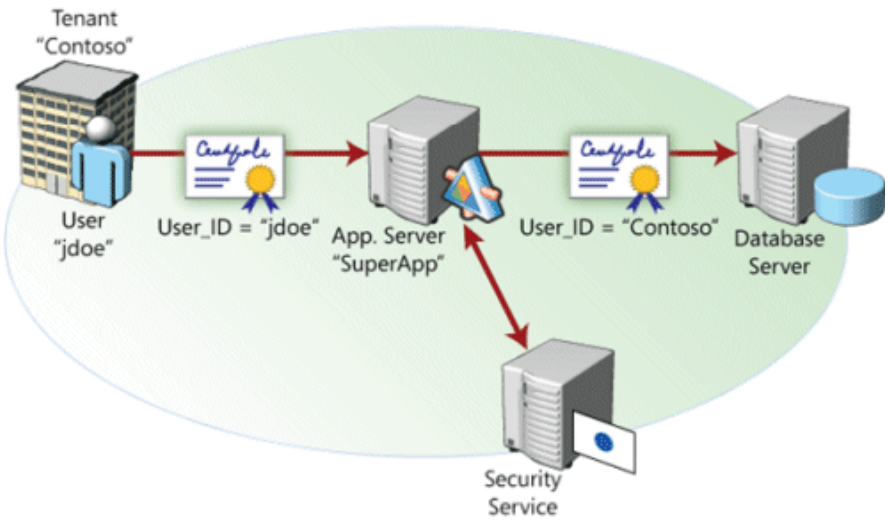

Figure 11 Illustration of the hybrid method (Source: Chong, *et al.,* 2006)

iv. Secure database tables

This approach can be used effectively with the separate database and separate schema approaches discussed in chapter 2. Under the separate database approach, access can be restricted on a database-wide level, to only allow the specific tenant access to their data. However, this can also be applied on a database table level, thus creating another layer of security. (Chong, *et al.,* 2006)

B – Filtering

Filtering is slightly more complex than the secured database table method, but it is an effective method to secure data in a shared schema database approach, where multiple tenants share the same database and tables. This can be achieved by using SQL views to provide individual tenants access to rows, while preventing others from accessing these rows. A SQL view is virtual table created by the results of a SELECT query. (Chong, *et al.,* 2006)

C – Encryption

The Microsoft Corporation (2008b) provides the following explanation for encryption: “The process of coding plaintext to create ciphertext is called encryption and the process of decoding ciphertext to produce the plaintext is called decryption. Modern systems of electronic cryptography use digital keys (bit strings) and mathematical algorithms (encryption algorithms) to encrypt and decrypt information.”

Encryption increases in importance as a SaaS application approaches the shared database architecture as discussed in chapter 2. Encryption can thus play an important role in securing sensitive database information where multiple tenants share database tables. However, as an encrypted column can not be indexed, some performance will have to be sacrificed for security. (Chong, *et al.,* 2006)

In the context of SaaS, tenant data encryption refers to the process of protecting data within a database, so that even if it is accessed by unauthorised persons, it would remain secure (Chong, *et al.,* 2006).

Chong, *et al* (2006) identifies three approaches to encryption that can be used in the context of SaaS:

i) Symmetric encryption;
ii) Asymmetric encryption; and
iii) Key wrapping.

i. Symmetric encryption

With this system data is encrypted and decrypted with the same key (Chong, *et al.,* 2006). This key is referred to as the symmetric key or secret key because it is kept as a shared secret between the sender and receiver of information. The confidentiality of the encrypted information may be compromised if the secret key became known to an unauthorised party. (Microsoft Corporation, 2008b)

ii. Asymmetric encryption

Asymmetric encryption is also referred to as public key cryptography. In this approach, two keys are used, the public and the private key. Data is encrypted using the widely distributed public key and can only be decrypted with the corresponding private key. (Chong, *et al.,* 2006)

This encryption method requires a significant amount of additional computing power as opposed to the symmetric approach. A strong asymmetric key pair can take up to a thousand times as long as a symmetric key pair of equal quality, to encrypt and decrypt. As a result of the additional computing power required, this form of encryption may not be feasible for a SaaS application where every piece of data has to be encrypted. (Chong, *et al.,* 2006)

iii. Key wrapping

As part of this process, three keys are generated for each tenant: a symmetric key, a public key and a private key. The symmetric key is used to encrypt the tenant's data. To add an additional layer of security, the symmetric key is encrypted using the public/private key pair. This ensures that the symmetric key is secured from unauthorised access. (Chong, *et al.,* 2006)

In the SaaS context, the application will use impersonation to access the tenant's database, using the tenant's security context. This in turn grants the application process access to the private key related to the specific tenant. While still impersonating the tenant, the application can use this private key to decrypt the symmetric key. With the decrypted symmetric key, the application is then able to read and write the tenant's data. (Chong, *et al.,* 2006)

Summary

The "defence in depth" principle maintains that not one single method for securing data is sufficient in its own right. However, when combined, permissions, filtering and encryption can provide a sufficient level of security to protect tenant data. For example, the first level of security would be implemented on the database level to prevent other users from accessing each other's data. Then, as a second level of defence, the actual data would be encrypted to ensure that data would be unreadable, even if access is gained. (Chong, *et al.,* 2006)

Security of data and information should be of major concern to the potential SaaS customer, as the physical storage thereof would take place outside of the control of the customer. It would be imperative to ensure that the security approach chosen by the SaaS vendor is sufficient to meet the needs of the specific customer.

4. Integration with existing systems

4.1 Introduction

Many companies have multiple applications that were never designed to work together due to incompatible architectures (Microsoft Corporation, 2008a). These applications can

include supply chain management applications, customer relationship management applications, business intelligence applications and many other types of applications.

The Microsoft Corporation (2008a) highlights certain important aspects which should be considered as part of the integration process:

- The integration process should be as non-invasive as possible: Any change to an existing system is a risk. It is therefore important to consider the impact of a change to the existing systems as part of responding to the needs of the users.
- The internal data structure of each application should be isolated: A change to one application's data structure could require multiple changes to other applications.
- To ensure efficiency in data exchange, the existing validation and data integrity checks in an application should be used.
- Direct access to an application's data may lead to a security violation.
- The availability of commercially available tools to perform the integration: Freely available tools may reduce the potential costs associated with a major integration exercise.

Integration is a major consideration for any customer considering a move to a SaaS application model. Careful consideration should therefore be given to the following three areas of integration:

- System integration (Carraro & Chong, 2006b);
- Identity integration (Carraro & Chong, 2006b); and
- Data integration (Microsoft Corporation, 2008a).

These three areas will be discussed in more detail, as specifically applicable to SaaS.

4.2 System integration

4.2.1 Definition

The Georgia State University (2008) defines system integration as "…the process by which different computing systems and software applications are linked together physically or functionally." The Georgia State University (2008) also states that system integration relates to "…the strategies and methods for blending a set of interdependent systems into

a functioning or unified whole, thereby enabling two or more applications to interact and exchange data seamlessly."

Sharpy (2008) states that a system is "A collection of sub-systems that communicate in some manner and are thus organized as a whole. The purpose of the system is different to the purposes of each individual sub-system and has an overall objective."

4.2.2 System integration under SaaS

With a SaaS application the actual processing and storage of data takes place outside of the local infrastructure of the company using SaaS. Thus, specific integration architecture has to be defined to bring this outside data into the local infrastructure. Additionally, the various infrastructure components (hosted internally or externally) should be able to the access the data that it requires, irrespective of where the data originates. (Carraro & Chong, 2006b)

Implementation

As part of the implementation of a SaaS application, data would have to be transferred between existing systems and the new application. This can be achieved in any of the following ways (Carraro & Chong, 2006b):

- "Bootstrapping" the SaaS application with existing data from current applications;
- Configuring the SaaS application to use on-premise data produced by existing systems as part of its operation; and
- Configuring existing on-premise applications to use data produced by the SaaS application, as part of its operation.

Integration of the SaaS application into the existing application infrastructure will require the creation of data dependencies that would require data to be moved and synchronised between the SaaS application and one or more existing applications. In this situation, an "Integration Broker" can be used to manage this data movement and application integration. (Carraro & Chong, 2006b)

The Integration Broker

Data can originate from many different sources, using incompatible protocols and formats. The function of the Integration Broker ("IB") is to convert and route data from various

sources and in various formats, to the correct destination and in the correct format for the intended application to use. (Carraro & Chong, 2006b)

Per the technical description of an IB, it “takes the form of a pipeline architecture to which you can add and remove modules that perform specific integration operations. Multiple logical pipelines can be used to process data travelling in different directions.” (Carraro & Chong, 2006b) This can be illustrated by the example of one pipeline that is responsible for integrating data from the Internet to local data sources and another for integrating local data with the SaaS data on the Internet. (Carraro & Chong, 2006b)

The various modules plugged into the pipeline determine the routing, processing and integration of the data with the destination data. Metadata can be used to configure the operation of each module. Carraro & Chong (2006b) identify the following common areas that can be configured:

- Security
    - A security module processes incoming data and performs functions such as authenticating, decrypting and examining the incoming data for security risks such as viruses.
- Validation
    - This module compares the incoming data to relevant schemas and can reject the data or pass it off to a data transformation module. The data transformation module will transform the data so that it can be used by the destination. This is particularly true with SaaS applications using a XML format and an on-premise application using a legacy data format. The process of data transformation firstly involves identifying the correct data format and validating the data against this. Secondly the data is converted to the target data format.
- Synchronisation workflow
    - The workflow module determines how data changes are transmitted to the destinations.
- Routing
    - The routing module determines the destination for each piece of data, either by using a simple end-to-end transmission or by determining the end destination by referencing the data itself.

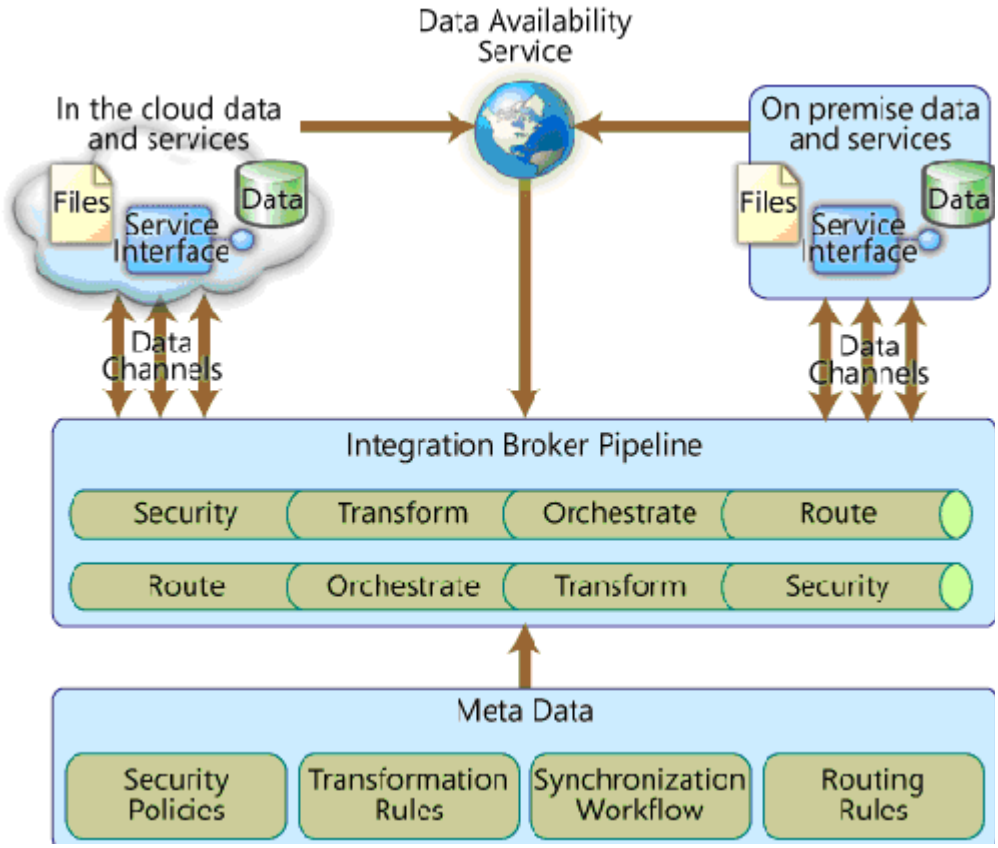

Figure 12 Illustration of an Integration broker system (Source: Carraro & Chong, 2006b)

4.3 Identity integration

4.3.1 Introduction

In the context of a SaaS application, the end-user should not be affected by the fact that the application is physically situated outside of the company's firewall. As a result, the end-user should not have to provide additional credentials when accessing the SaaS application. (Carraro & Chong, 2006b)

Single sign-on enables users to access an application without having to present a separate set of credentials. In other words a user can log into the operating system once at start-up and can thereafter access various applications without having to present a different username and password. (Carraro & Chong, 2006b)

4.3.2 Technical set-up of SSO in a SaaS environment

In a SaaS model SSO authentication occurs via the use of a federation server located inside the premises of the customer, which interfaces directly with the customer's own

directory services. This federation server has a trust relationship with a federation server located at the SaaS vendor's premises. Whenever a user attempts to gain access to the SaaS application, the local federation server will determine whether the user has the authority to use the system. A security token is then issued, which the SaaS vendor's federation server accepts and access is granted to the end-user. (Carraro & Chong, 2006b)

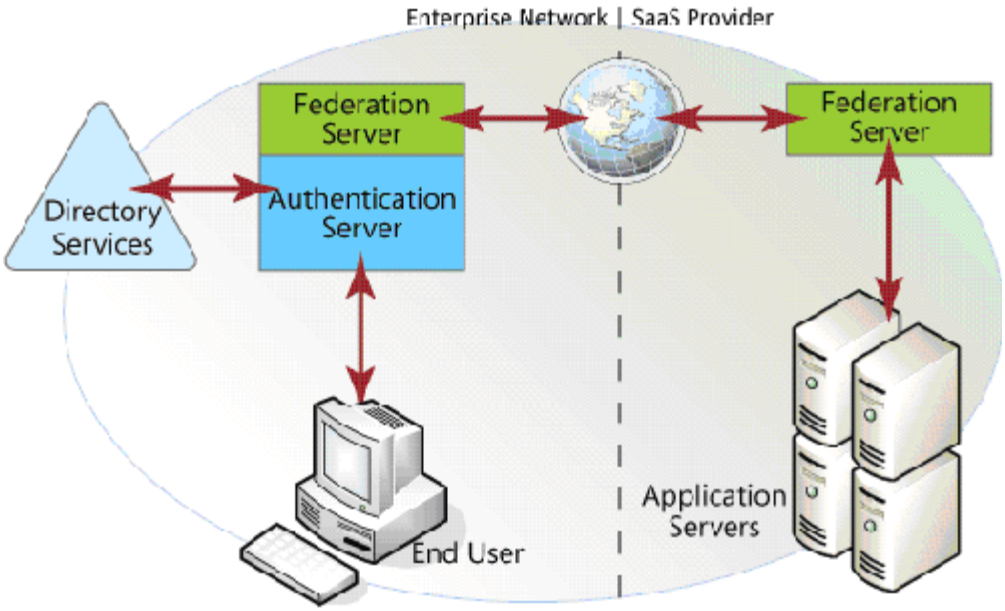

Figure 13 Illustration of a single sign-on set-up in a SaaS application (Source: Carraro & Chong, 2006b)

4.3.3 Advantages of SSO in a SaaS application model

Carraro & Chong (2006b) lists the following benefits that the SSO architecture has for the company subscribing to a SaaS application:

- Users have fewer passwords to keep track off. This reduces the risk that passwords may become known. (Opengroup, 2005)
- Fewer credentials to manage. IT support staff have less user credentials to administer and maintain, thus reducing the time spent by IT staff on user administration. (Opengroup, 2005)
- SSO enables identity integration. Existing user profiles and policies can be used to control access to SaaS applications.

4.4 Data integration

4.4.1 Introduction

In the case where a SaaS application has to access a database of another application on a local server or vice versa, careful consideration should be given to the integration technique that is to be used to enable this sharing of data between diverse applications.

The basic principle behind data integration is explained by the Microsoft Corporation (2008a) as the integration of “applications at the logical data layer by allowing the data in one application (the source) to be accessed by other applications (the target).”

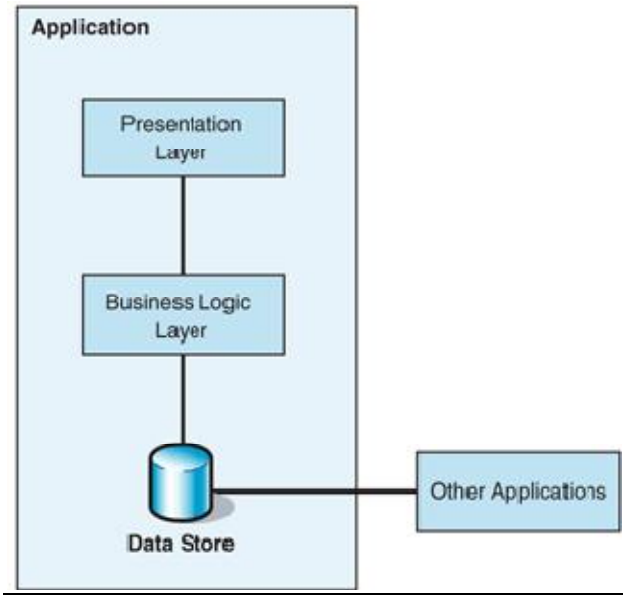

Figure 14 Illustration of data integration at the logical data layer (Source: Microsoft Corporation, 2008a)

4.4.2 Considerations

Various considerations have to be taken into account as part of the data integration decision. These are briefly discussed below.

General

The following considerations are discussed by the Microsoft Corporation (2008a):

- Latency tolerance: Latency relates to the delay between updates to data that is shared amongst applications. This delay may result in a situation where one

application has access to data that is more up to date than that of another application.

- Granularity: Updating a larger amount of data at one time is generally more efficient than updating each small change on its own. However, a thorough understanding of the relationships between data entities is necessary to implement this correctly.
- Master/subordinate relationships: If updates are only made to one application's data, transferring the updated data is relatively simple. However, if multiple applications are involved, the process becomes complex.
- Synchronisation logic versus latency: Sharing a single database may result in excessive network latency, where applications are geographically dispersed. To overcome this, distributed databases containing copies of the related data can be created. However, this creates additional complexity in the form of synchronisation and replication set-up and maintenance.

Security

Data integration creates potential security issues that have to be considered carefully. The main considerations, as set out by the Microsoft Corporation (2008a), are:

- Coarse grained security: Due to the fact that data integration bypasses application logic, it will also bypass any application security rules. Databases manage access privileges at a table level, whereas applications tend to manage access privileges at an object level. As the data integration bypasses the application security rules, it will also bypass the access restrictions on an object level, therefore inadvertently granting access to the entire database table.
- Privacy policies may not be enforced: Many corporate databases have to adhere to strict corporate and legal privacy guidelines. Directly accessing the database through data integration may violate these guidelines, as it is difficult to control the use of the retrieved data. However, "functional integration" can control access to sensitive data via queries, as to not expose sensitive data.
- Data encryption within a database: Encrypted data can not be integrated with other data, unless the integration tool obtains access to the encryption key (refer to section 4 above for more details regarding encryption). However, providing access to the encryption key may lead to a security violation, unless the key is appropriately protected.

Data availability

A data availability service enables the Integration Broker to identify when updates have been made to data elements. Synchronising data is the process of transferring new or updated data from the destination to the source at regular intervals or when a certain event takes place. (Carraro & Chong, 2006b)

According to Carraro & Chong (2006b), there are three basic methods that are used to trigger data synchronisation between a local source and a SaaS application:

- Polling: The data source queries the other data entities for updates at regular intervals.
- Push: Push is the opposite of Polling. The data entity responsible for the data change, communicates this to the data destination on regular intervals or whenever a change takes place.
- Publish and subscribe: This is a hybrid approach which combines aspects of polling and pushing. Whenever a change is made to the data source an event is published, to which the data sink can subscribe.

The choice of approach will depend on various factors as identified by Carraro & Chong (2006b), including:

- If data changes have to be reflected at or near real time;
- The number of data sinks to be updated with a data change; and
- The type and amount of data to be updated at any given point in time.

4.4.3 Data integration patterns

Introduction

To be able to connect applications at the logical data layer, one of the following methods can be used (Microsoft Corporation, 2008a):

- Shared database;
- Maintain data copies; or
- File transfer.

Shared database

With this model, all applications that are integrated, read data directly from the same database. This method may be more intrusive to implement under the SaaS application

model, as it requires all applications to be able to use the same database schema. However, this can be overcome through the use of a specialised module within the Integration Broker. (Microsoft Corporation, 2008a)

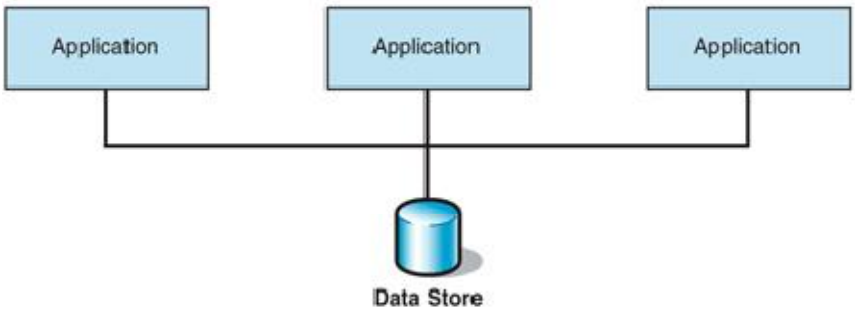

Figure 15 Illustration of shared database approach (Source: Microsoft Corporation, 2008a)

Maintain data copies

Multiple copies of the central database are made for each application that requires access to the database. Each application will therefore have its own dedicated data store. This adds the complexity of ensuring that the data stores are synchronised. As a result, there is a risk that, due to latency, data stores may be out of synch at any given point in time. (Microsoft Corporation, 2008a)

For the SaaS application model, this would imply that both the SaaS vendor and the customer would have to maintain a separate database. Also, if large amounts of data have to be synchronised between databases, additional network bandwidth would be required. These factors will lead to increased costs and administrative tasks for the customer, which would negate some of the costs savings of using a SaaS model in the first place. (Microsoft Corporation, 2008a)

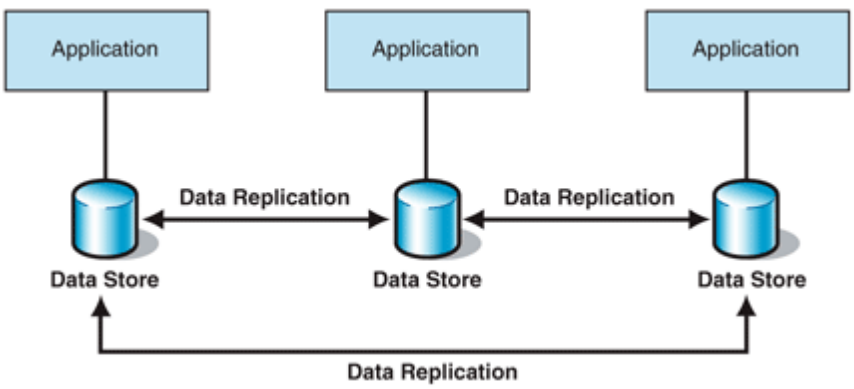

Figure 16 Illustration of “data copies” approach (Source: Microsoft Corporation, 2008a)

File transfer

Under this pattern, one application produces a file and transfers it to other applications for processing. This method is often simple to implement, as most operating systems recognise files as a universal unit of storage. However, two applications can lose synchronisation as they both independently update the file. (Microsoft Corporation, 2008a)

This approach may not be suitable for SaaS due to the amount of bandwidth that may be required to transfer files between applications.

Choice between different approaches

The Microsoft Corporation (2008a) lists the following factors to consider as part of the choice between the above models:

- The level of tolerance for data that is stale;
- Performance;
- Complexity; and
- Infrastructure and integration support.

Table 2 Factors to consider per method (summarised from the information above)

Consideration	Shared database	Maintain data copies	File transfer
Creates stale data	No	Yes	Yes
High levels of performance	No	Yes	No
Less complex	No	Yes	Yes
Requires high levels of support	Yes	Yes	No
Creates high levels of network latency	Yes	Yes	Yes
Application and database located in same data centre	Yes	No	No

4.5 Summary

Integration of systems, identity and data, is a critical component in the architecture strategy to incorporate SaaS successfully, as a fully participating member of a service-centric IT infrastructure. Without effective and efficient integration, a SaaS application cannot function as a successful part of a company. (Carraro & Chong, 2006b) As the development of SaaS progresses, more and more SaaS application “suites” are being

developed, which would do away with the need for integration of various systems. However, these application suits are still in their infancy and are as yet still to be proven effective. (Gruman, 2007a)

In section five, scalability will be discussed as it is important for the potential SaaS customer to consider if the SaaS application will be able to grow with its business.

5. Scalability

Scalability refers to the ability to increase the numbers of users or the amount of data that an application can support. This is even more important in a SaaS environment where multiple end-users have to be supported per customer, multiplied by the number of customers supported. (Carraro & Chong, 2006a)

5.1 Scaling applications

To be able to support more users, applications can be scaled up or scaled out. By scaling up, an application is moved to a more powerful server, with more storage capacity. When scaling out, the application is re-deployed to run over multiple servers. Scaling up is the simplest method to increase performance of a small application that does not support a large number of concurrent users. However, in the SaaS model scaling out is the best solution, as this fits in with the maturity levels of SaaS as discussed in chapter two. With this model, a well designed SaaS application can be scaled out to a large number of servers, running identical instances of the SaaS application. (Carraro & Chong, 2006a)

5.2 Scaling data

With a mature SaaS application, one database is used to serve thousands of end-users. This approach can quite easily suffer from poor performance. A simple solution to poor performance is to partition the database into smaller portions, to enable queries and searches to be carried out more efficiently. However, as the number of users increases, so will the amount of data to be stored. It is therefore very important to have a dynamic partitioning strategy in place to provide for this growth. (Carraro & Chong, 2006a)

Service level agreements should be considered carefully when deciding on a move to a SaaS application, as this will impact the ability of the SaaS customer to hold the SaaS vendor to specified service levels. This is discussed in section six below.

6. Service level agreements

6.1 Definition and applicability to SaaS

A service level agreement ("SLA") is "…a document which defines the relationship between two parties: the provider and the recipient." (Service Level Agreement Zone, 2002) SLA's are legally binding agreements, and the failing to meet the terms of the agreement may lead to significant losses for the service provider, in this case the SaaS vendor (Carraro & Chong, 2006a).

As the SaaS application model involves a transfer of responsibility from a company to an external supplier, SLA's can be used to manage the levels of service expected by the SaaS customer.

6.2 Impact on potential SaaS customers

Currently, many SaaS vendors do not have formal SLA's in place with their customers. Customers may have little recourse if the SLA is not in writing (Wailgum, 2008). This is particularly true in the case of the middle market customers, where up to 85% of the SaaS application do not have any form of SLA in place with the vendors. SaaS vendors supplying services to large enterprises are more likely to have formal SLA's in place. (Gruman, 2007a)

<u>Common SLA shortcomings</u>

It is important for the potential SaaS customer to consider the following common contracting risks associated with SaaS applications. The customer should ensure that the SLA with the SaaS vendor contains the necessary protection against these contracting risks (Wailgum, 2008):

i) Hidden cost drivers
ii) Unexpected downtime

iii) Declines in levels of customer support

iv) Obscure disaster recovery procedures

v) Abdicating all support to the vendor

vi) Changes in ownership of the SaaS vendor

i) Hidden cost drivers

As the amount of data and users increase over time, so will the overall cost of the deployment of the SaaS application. (Wailgum, 2008) The SLA should specify any costs associated with the duties of the SaaS vendor, which is not covered in the recurring subscription fee.

ii) Unexpected downtime

Typically, SaaS SLA's will include a 99.5% uptime guarantee. However, this normally does not include planned downtime for maintenance. In addition, many SaaS customers do not keep track of the actual uptime delivered by the vendor. As a result, users are not able to enforce the SLA and the requisite penalties involved for unplanned or un-communicated downtime. (Wailgum, 2008) Typical clauses that should be included in a SLA with a SaaS vendor would include (Finch, 2008):

- Uptime percentage guarantees, including discounts for unplanned downtime;
- Advance system maintenance notifications; and
- Outage notifications, including a full description of the problem and a plan for the resolution thereof.

iii) Declines in levels of customer support

Most users are satisfied with the initial levels of customer support. However, over time it can become increasingly difficult to get support from senior level developers within the SaaS vendor. (Wailgum, 2008) Typical clauses that should be included in a SLA with a SaaS vendor would include (Finch, 2008):

- Network access protection policies and procedures;
- Technical support services and procedures; and
- Code fix and upgrade procedures.

iv) Obscure disaster recovery procedures

Many SaaS users do not understand the security and disaster recovery clauses in their SLA's with SaaS vendors. It would be advisable for the potential SaaS customer to

perform a due diligence around the disaster recovery procedures of a SaaS vendor before entering into an agreement. (Wailgum, 2008) Typical clauses that should be included in a SLA with a SaaS vendor would include (Finch, 2008):

- Documented disaster recovery and business continuity plans;
- Data backup procedures; and
- Restore procedures.

v) Abdicating all support to the vendor

Even thought this is one of the main attractions of moving to a SaaS model, the accountability for the IT systems, in the eyes of the end-users, still lies with the internal IT department. It is therefore important for the in-house IT department to ensure that the various responsibilities for the SaaS application are clearly defined. (Wailgum, 2008) Typical clauses that should be included in a SLA with a SaaS vendor would include (Finch, 2008):

- Regulatory considerations for certain data types;
- Restricting vendor from decrypting or viewing company data;
- Protection of company data on vendor's mobile devices;
- Physical and logical security procedures;
- Controls and policies for media and devices, to protect data;
- Data transmission security policies and procedures; and
- System and security monitoring.

vi) Changes in ownership of the SaaS vendor

The SaaS customer should ensure that they are comfortable with the fact that the SaaS vendor might become part of another company over time. (Wailgum, 2008) Typical clauses that should be included in a SLA with a SaaS vendor would include (Finch, 2008):

- Procedures for returning or destroying data;
- Restricting ownership of company data to the company;
- Code escrow provisions;

7. Summary

Major considerations in the decision to move to SaaS include:

- Availability of services and data;
- Security and privacy of data;
- Integration with existing systems;
- Scalability; and
- Service level agreements.

These issues were discussed in detail in this chapter and should be considered by the potential SaaS customer, as part of the decision to move to a SaaS software delivery model. However, probably the most important factor to consider as part of the move to SaaS is the cost impact on both the SaaS vendor and the customer. Costing will be discussed in more detail as part of chapter five.

Chapter 5 – Costing

1. Introduction

Any company would easily add software functionality without additional hardware infrastructure. However, no company would add hardware infrastructure without a need for additional software. Additionally, with most on-premise based software applications, the majority of the budget is spent on infrastructure and staff costs, with a minority of the budget left for the actual software application. (Carraro & Chong, 2006a)

For any software implementation, the key cost components are the cost of the actual software application, the hardware needed to support the application and the staff costs to deploy, maintain, support and upgrade the application. Under a traditional on-premise application model, the cost of the software for the customer is limited to the perpetual licensing fee. However, the additional hardware and personnel costs would also be carried by the customer. (Sysmans, 2006)

More than 75% of a typical IT budget can be spent just on the maintenance and operation of existing on-premise based software applications. Of the maintenance costs, personnel costs can be as high as 70%. Most companies spend up to four times the cost of their initial licences on managing these applications. The cost of the initial purchase of the application can be as low as only 5% of the total cost to own an on-premise based application. (Sysmans, 2006)

With a SaaS application, the implementation costs are normally between 25% and 40% of the costs associated with the implementation of an on-premise based application. Additionally, the cost of hardware infrastructure can be brought close to zero. (Lashar, 2008)

SaaS applications are normally “rented” on a subscription basis, which can be based on the number of users per month, or usage per month. There are no up-front licensing fees payable under a SaaS model as the monthly subscription fee covers maintenance, upgrades and support. (Gruman, 2007b) As an added bonus, no additional hardware infrastructure is needed to be able to support the SaaS application (Businessweek, 2006).

Typical SaaS subscriptions are significantly higher than annual maintenance fees payable to traditional on-premise based application vendors. Over the medium to long term the total cost of these subscription fees may approach and even exceed the payments to an on-premise vendor. (Lashar, 2008) Therefore, the most difficult part of determining whether to move to SaaS is the total cost of ownership ("TCO") calculation. This is due to the fact that the TCO is not just the cost of new licences, but also the cost of integration, support, training and operations. However, many companies feel that not having to manage the application is often worth the extra cost. (Gruman, 2007b)

In this chapter the various cost components and licensing models of SaaS will be discussed. A summary table of the cost considerations between the traditional on-premise based software delivery model and SaaS model will be presented at the end of this chapter.

2. Cost components of software applications and the impact of SaaS

2.1 Introduction

According to Carraro & Chong (2006a), the typical IT budget can be divided into three broad areas:

- Software: The actual applications and information that a company requires as part of their data processing needs.
- Hardware: The physical computing infrastructure in the form of servers, desktop computers, network connections, etc.
- Professional services: The personnel required to operate and maintain the software and hardware infrastructure. This includes technical staff, consultants and representatives from software vendors.

Software is the most directly involved in the management of information, which is the ultimate objective of any IT department. Hardware and professional services are seen as a means to an end, even though they form important components of any IT environment. (Carraro & Chong, 2006a)

2.2 Cost comparison between traditional and SaaS applications

2.2.1 On-premise based applications

In a typical on-premise application's IT budget, the software budget is normally spent on the up-front licensing fee of the application as well as any periodical maintenance fees. The hardware budget is spent on servers, desktop computers and network components. The professional services budget is used to pay for support staff and IT consultants to deploy and maintain the application and to customise the application to the needs of the company. (Carraro & Chong, 2006a)

Figure 17 Budget split for a traditional on-premise application (Source: Carraro & Chong, 2006a)

Figure 17 illustrates the split of a typical IT budget for an on-premise based software application. It should be noted that the proportions shown in this diagram are for illustrative purposes only and they are not intended to represent any specific allocation of resources, as this may vary from business to business.

2.2.2 SaaS applications

With a SaaS application, the SaaS vendor hosts the application and data centrally at its own location. All hardware and software support is provided by the SaaS vendor's dedicated support staff. As a result, the customer does not have to maintain and support the software and also does not have to acquire additional hardware to run the application. In addition, a SaaS application delivered over the Web, or via another form of thin client, requires significantly less computing power and can therefore lead to an extended desktop technology life cycle. (Carraro & Chong, 2006a)

The end result of this is that a larger percentage of the overall IT budget is available to be spent on pure software functionality. This is typically done in the form of subscription fees to the SaaS vendor. However, many may argue that a portion of the subscription fees paid to the SaaS vendor has to go toward hardware and professional services for the vendor. Due to the multi-tenant architecture of a mature SaaS application and the related economies of scale, a SaaS vendor can provide these services at a reduced cost to customers. (Carraro & Chong, 2006a)

For example, a SaaS application hosted on a load balanced farm of five servers may be able to serve 50 customers from a single instance of the SaaS application. In essence each customer would then only be responsible for a tenth of the cost of the servers. In contrast, a similar application hosted on-premise, may require the customer to dedicate an entire server to the application. From this it is clear that the SaaS application model may have significant cost advantages over the traditional application delivery approach. (Carraro & Chong, 2006a)

The potential costs savings will therefore continue to increase as more and more SaaS vendors develop multi-tenant capable applications. This in turn leads to higher quality software applications at lower costs. Therefore, even after taking into account the hardware and software costs of the SaaS vendor, customers can still achieve costs savings when compared to an on-premise based application. As a result greater software functionality can be obtained from the same IT budget. (Carraro & Chong, 2006a)

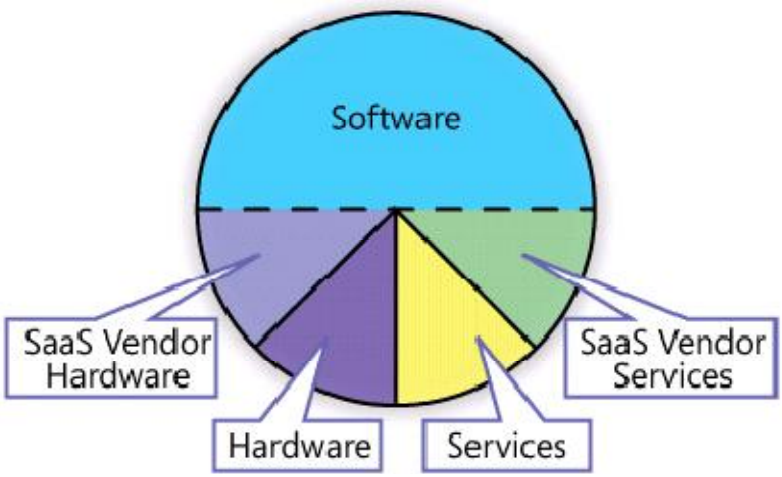

Figure 18 Budget split for a SaaS application (Source: Carraro & Chong, 2006a)

Figure 18 above details a typical SaaS IT budget, taking into account the portion of the fees payable to the SaaS vendor which is used for the SaaS vendor's hardware and services. This is done to provide a more accurate representation of the budget allocation in a SaaS model. When compared to figure 17, it is clear that more of the IT budget is available for the customer to spend on software. This is due to the fact that substantially less is spent on hardware and IT services, as these costs are carried mostly by the SaaS vendor. (Carraro & Chong, 2006a) It should be noted that the proportions shown in this diagram are for illustrative purposes only and they are not intended to represent any specific allocation of resources, as this may vary from business to business.

Detail cost components

The following detailed cost components, normally applicable to customers, are identified by Sysmans (2006), and further expand on the three IT budget areas as mentioned earlier:

1. Capital expenses;
2. Design and deployment costs;
3. Ongoing infrastructure costs;
4. Ongoing operations, training and support costs; and
5. Intangible costs.

By using these costs components, the traditional software application model and the SaaS model can be compared as per table four below.

Table 3 Comparison between traditional software application and SaaS application costs

Cost component	Traditional Software	SaaS application model
Capital expenses	For traditional software the typical capital expense components will include (Sysmans, 2006): • Software and hardware; • Supplies; • Network infrastructure; • Monitoring and testing tools; • Security products; • Upgrades to other systems; and • An initial license fee. These items all represent additional, up-front cash commitments that would have to be borne by the customer.	With a SaaS model there are no up-front license or infrastructure costs. (Gruman, 2007a) Customers pay on a per-month, per-use or per-transaction basis, in the form of a subscription (Software & Information Industry Association, 2000b). This recurring subscription fee would include the costs for the SaaS vendor to provide hardware and services to the customers. In essence this would reduce the up-front capital outlay needed for a customer, when implementing a new software application.

Cost component	Traditional Software	SaaS application model
Design and deployment costs	A large software implementation is normally accompanied by significant design, deployment and consulting costs (Software & Information Industry Association, 2000b). Staff and contract personnel are needed to research, test and tune the new and existing systems. All of these activities represent significant outlays in terms of personnel costs. (Sysmans, 2006) Significant, additional design and deployment costs would be incurred in the form of personnel and consultant costs as mentioned above. Thus increasing the cost of a traditional software installation.	With a SaaS application there are no up-front development and equipment costs (Gruman, 2007b). There is also no on-site deployment as with a traditional software application (Gruman, 2007a). Thus reducing costs. A SaaS application can be put into operation with a fraction of the effort needed for a traditional software application. This is an important consideration when the opportunity cost of the implementation time of the application is high (Sysmans, 2006). Thus, as less design and deployment effort is needed for a SaaS application, costs are kept low.
Ongoing infrastructure costs	Included in this component is the following costs (Sysmans, 2006): • Network monitoring and management tools; • Additional network equipment and bandwidth; • Annual system maintenance fees and upgrades; • Redundant systems; • Hardware repair and replacement; and • Recurring environmental costs including high-availability facilities and power consumption. Even though these costs are spread out across the lifetime of the application, they have to be taken into account in the calculation of total cost of ownership (Sysmans, 2006).	For a SaaS application, software maintenance, operational costs as well as software upgrade costs are included in the subscription fee paid to the SaaS vendor (Gruman, 2007b). Other than additional Internet bandwidth, there are almost no additional incremental costs associated with SaaS applications. (Sysmans, 2006). As a result the ongoing infrastructure costs, that are associated with the SaaS application is minimal.
Ongoing operations, training and support costs	The in-house IT department is normally responsible for ongoing operations, training and support (Sysmans, 2006) As a result the personnel costs associated with the application would increase the TCO of the traditional on-premise application.	For a SaaS application, software maintenance, operational costs as well as software upgrade costs are included in the subscription fee paid to the SaaS vendor (Gruman, 2007b). SaaS vendors normally also provide ongoing training and support for their applications. (Sysmans, 2006). The only real responsibility of the in-house IT department is to ensure that the required firewall ports are open and that there is enough Internet capacity to enable the efficient use of the application (Sysmans, 2006).

Cost component	Traditional Software	SaaS application model
		As a result the costs associated with ongoing operations, training and support is restricted to the recurring subscription fee payable to the SaaS vendor.
Intangible costs	These issues are common between both the traditional software approach as well as the SaaS model. Intangible costs may be hard to quantify in terms of a total cost of ownership calculation. However, these issues still represent significant considerations in the choice between application models. The following are some of the common items that should be considered: • Reliability and availability Unavailability of a system leads to lost opportunities and employee time. It is therefore important to consider the SLA in place with the SaaS provider and compare this to the internal SLA with the IT department. (Gruman, 2007a) • Integration The more complex and diverse the systems are that have to be integrated, the higher the development cost would be. (Gruman, 2007a) • Extensibility The more extensive the customisation needed to a software application, the more expensive the implementation would be. (Gruman, 2007a; Sysmans, 2006) • Security The costs associated with a breach of security related to confidential business information can be significant. Careful consideration should be paid to the security policies of the SaaS vendor and how they compare to the internal security policies. (Sysmans, 2006; Gruman, 2007a) • Scalability It is important to plan for future growth in the use of the application. As a result it should be considered if the SaaS vendor would be able to support growth and the comparable costs associated with an internal application. (Sysmans, 2006) • Capacity It may be difficult to predict the correct levels of infrastructure needed to ensure that the performance level of the application is acceptable. With a SaaS application, this is the responsibility of the vendor. (Sysmans, 2006) • Opportunity costs An in-house implementation requires extensive human resource and capital expenditure, which could have been used for other activities more closely related to revenue generation. (Carraro & Chong, 2006b)	

3. Summary

As is evident from the above the most significant cost of an on-premise based application is the ongoing human resources required to support and maintain the application. These costs are normally not included in the initial estimate of the total cost of ownership (“TCO”), but can be between 50% and 85% of the TCO. In contrast, the most significant factor in the TCO calculation of the SaaS application is the ongoing subscription fees payable to the vendor. These fees include all the related costs for monitoring and supporting the application which are normally included in the costs for deployment of a SaaS application. (Sysmans, 2006)

It is of utmost importance to ensure that the decision of which software application to acquire, is not based on short term considerations only. This can be an expensive mistake, even for a relatively inexpensive SaaS application. (Lashar, 2008)

In the following chapter the main differences between the SaaS application model and the traditional on-premise based application model is summarised. In addition to this a total costs of ownership (“TCO”) calculation framework is presented to aid the potential SaaS customer in the decision to move to the SaaS application delivery model.

Chapter 6 – Total cost of ownership framework based on SaaS considerations

1. Introduction

The choice between SaaS and traditional on-premise based software applications should include more considerations than cost trade-offs alone. Additional areas to consider include business benefits, flexibility and associated risks. In general, management should obtain a deep understanding of all areas that will be affected by the acquisition and implementation of the application. (Wailgum, 2008)

2. Summary comparison between SaaS and the traditional on-premise based application delivery model

It is of utmost importance for the potential SaaS customer to fully understand the differences between SaaS and the traditional on-premise based applications, to be able to make an informed decision relating to SaaS as an alternative. Table 4 illustrates the differences between SaaS and the traditional on-premise based software application delivery model; based on major differentiating characteristics. This table was compiled from the information presented in the previous chapters of this study.

Table 4 Comparison between software delivery models

Description	On-premise	SaaS
Location	The application is hosted in each customer's own data centre (Carraro & Chong, 2006b).	The application is hosted centrally at the SaaS vendor's premises. Refer to chapter 2 for details of the definition of SaaS.
Licensing	In perpetuity, up-front plus a yearly maintenance fee. However, the costs may increase if additional end-users require access to the application. (Carraro & Chong, 2006b)	Billed on a usage or time-based model. No up-front fees are applicable (Carraro & Chong, 2006b). Costs may increase as the number of end-users increase for each tenant (Sysmans, 2006).
Architecture	Single tenant, client-server applications (Sysmans, 2006).	Single tenant or multi tenant, net-native applications which are accessed through a thin client or web-browser (Bleicher, 2006).

Description	On-premise	SaaS
Management	Support and maintenance are performed by the in-house IT department. In certain cases, some of these activities are outsourced to third party service providers. (Carraro & Chong, 2006b)	All management tasks are performed by the SaaS vendor. The SaaS vendor will employ dedicated support teams to assist customers. (Sysmans, 2006)
Integration	As all applications are hosted locally, complex, real-time integration is possible (CRM Landmark, 2007).	Integration with existing applications is complex and requires additional development. Refer to chapter 4 for a detailed discussion surrounding integration.
Upgrades	Upgrades are done periodically and require considerable planning as to not disrupt the normal business operations. These upgrades are performed by staff from the in-house IT department and in the case of complex applications, the software vendor. (Sysmans, 2006)	All updates are performed seamlessly in the background by the SaaS provider. No scheduled down-time is necessary to deploy these updates and the user will simply have access to the updates the next time they log into the application. Refer to chapter 3 for details.
Capital expenditure	Implementation of the application requires a capital investment in hardware, backup services and network infrastructure (Sysmans, 2006).	This model does not require additional hardware infrastructure to be acquired as exiting hardware and network connections are used. All hardware is provided by the SaaS vendor. The cost of the SaaS vendor's hardware infrastructure will form part of the subscription fee payable. Refer to chapter 5 for details.
Customisation	The applications can be customised as far as the license agreement with the software vendor allows. Customisation can be performed by the IT staff or the software vendor. (Carraro & Chong, 2006b)	In a single tenant implementation of SaaS, customisation can be performed for the customer by the SaaS provider. However, with a multi-tenant, single instance implementation, customisation is more complex and is normally replaced by configurable metadata for each tenant. Refer to chapter 2.
Security	The customer is responsible for security – physical and logical (Sysmans, 2006).	The SaaS provider assumes total responsibility for security of customers' data. (Sysmans, 2006)
End-user training and support	The customer is responsible to employ sufficiently knowledgeable staff to provide ongoing training as well as support. This function may also be outsourced in certain instances. (Sysmans, 2006)	The SaaS vendor assumes all training and support activities for the customers. The cost of these activities is included in the recurring subscription fees. (Sysmans, 2006)

Description	On-premise	SaaS
Performance	Typically provides a higher level of performance than SaaS, as the application is hosted in the customer's own data centre and can be accessed over the customer's local area network. (Carraro & Chong, 2006b)	Depending on the physical and logical connection to the customer, the performance of the application can vary. However, it is unlikely that performance will be equal or better than an on-premise solution. (Software & Information Industry Association, 2000c; Carraro & Chong, 2006b)

The following table represents certain guidelines, which may be useful to take into account in the choice between SaaS and traditional software application models. However, it is recommended that the full TCO calculation be completed as detailed in section 3 below.

Table 5 SaaS vs. Traditional software (Summarised from: CRM Landmark, 2007)

Consider SaaS if the following is applicable to your company	**Consider a traditional software application if the following is applicable to your company**
• The company's expense budget is larger than the capital budget. • Limited IT and technical support is available in-house. • The company has a distributed workforce. • It is a sales or service oriented business. • The company wants to implement standardised industry processes. • The company uses standard data structures • Integration of the application will not be complex. • Non-proprietary data is used. • The company requires variable pricing and low up-front costs.	• The software will require a great deal of customisation to comply with business processes. • The application is closely related to the core function of the business. • Large investments in internal infrastructure for IT have been made. • Sufficient internal IT resources are available. • Regulatory requirements prohibit the distribution of data outside the company's firewall. • Specialised data structures are required to meet business requirements. • Complex, real-time integration is required. • The company can afford the up-front capital investment and fixed costs. • Data that is generated is highly sensitive.

3. Total costs of ownership (“TCO”) calculation framework

To be able to make an informed decision between the traditional software application models and the SaaS model, the TCO should be considered with all applicable components included in the calculation thereof. This is represented by table eight, which has been adapted from the TCO calculator presented by Sysmans (2006). The purpose of this table is to provide a framework of the main considerations to be taken into account in the decision for the adoption of SaaS.

Table 6 Total cost of ownership calculation (Adapted from: Sysmans, 2006)

	Initial Costs		**Ongoing costs**							
	Year 0		**Year 1**		**Year 2**		**Year 3**		**Year n**	
Cost Component	**Traditional (T)**	**SaaS (S)**	**T**	**S**	**T**	**S**	**T**	**S**	**T**	**S**
Deployment										
Hardware										
• Servers										
• Network infrastructure										
Connectivity (considering expected levels of data transmission)										
• Physical connections										
• Logical connections										
• Additional bandwidth										
Software and license costs										
People costs										
• Internal IT staff										
• Consultants										
Integration with existing systems										
Customisation										
Maintenance and operations										
Ongoing people costs										
• IT and helpdesk staff										
• Consultants										
• End user training										
Monitoring and security										
Scheduled maintenance										
Design and engineering										
Data backup and recovery										
Unscheduled downtime and disaster recovery										
General administration										
External assurance										
Service level agreements										

	Initial Costs		Ongoing costs							
	Year 0		Year 1		Year 2		Year 3		Year n	
Cost Component	**Traditional (T)**	**SaaS (S)**	**T**	**S**	**T**	**S**	**T**	**S**	**T**	**S**
Intangible costs										
Security of data and information										
Functionality of the application and business requirements										
Reputation of vendor										
Ownership of data										
Performance levels over time										
Applicable laws and regulations										
Data storage requirements over time										
Availability and reliability of application										
Extensibility										
Growth potential										
Opportunity cost of resources involved										
TOTAL COSTS										
TOTAL COST AFTER YEAR n										
TOTAL COST OF OWNERSHIP										

Chapter 7 – Conclusion

In its simplest form, SaaS enables IT departments to move away from the role as application developers to application users. This, in turn, frees up valuable in-house resources that can be re-allocated to other more critical business areas. In essence, IT staff can play a more strategic role in achieving the company's main objectives. (Software & Information Industry Association, 2001)

With SaaS, companies can first choose their IT priorities and then choose the most appropriate application solution from a growing number of SaaS applications. Additionally, these applications have a significantly reduced implementation time and are hosted on the generally superior infrastructure of the SaaS vendor. Fundamentally, this can give management an opportunity to focus on the value added, core competencies of the company, while reducing costs and increasing efficiency. (Software & Information Industry Association, 2001) However, price is not everything. If it is the company's strategy to use IT as a core competitive advantage, then it might be better to use an on-premise solution (Schwartz, 2007).

Even though SaaS may be a valuable resource to smaller companies, it may not necessarily provide in all the needs of larger enterprises. While SaaS may still play a role in larger enterprises, it may only be a small role due to its limited customisability. (Gruman, 2007a) This is especially true when applying the 80-20 rule to software applications: 80% of a company's processes are standardised and can easily be outsourced via SaaS, however the remaining 20% is where the company differentiates itself and it will be difficult to apply to a SaaS model to these processes (Schwartz, 2005).

From the foregoing discussions it is evident that many considerations need to be taken into account before making the decision to move to the SaaS application delivery model. This includes the following main considerations:

- Choice of data architecture;
- Physical and logical connection means;
- User requirements and customisation;
- Availability of service and data;
- Security and privacy of data;
- Integration with existing systems;

- Scalability;
- Cost implications; and
- Service level agreements.

It was identified that there are no clear and comprehensive guidelines or frameworks, detailing all the relevant considerations to be taken into account in the decision to adopt SaaS. A need was thus identified for a holistic view of the considerations for the adoption of SaaS, which was addressed as part of this study through a detailed discussion of the aspects and considerations as listed above. Furthermore, as part of this document the main considerations were summarised and presented in table format, included in chapter six, to aid in this decision making process. This framework is based on a “Total Cost to Company” calculation to assist in the decision between a traditional software application and SaaS.

The value of this framework is that it enables both the potential SaaS customer to make an informed decision, based on monetary implications, as to the viability of SaaS as an alternative to an on-premise based software application. It should however be noted that even though the cost of a SaaS application may be less than that of a traditional software application in monetary terms, certain companies would still prefer to use a traditional on-premise based application, due to fears surrounding security, service levels and availability.

Careful consideration should be taken as to what the main priorities, objectives and goals of a company are, as it is impossible to apply a “one-size-fits-all” approach to SaaS.

References

Bleicher, P. 2006. Solutions Delivered, Not Installed. *Applied Clinical Trials.* [Online]. June, 15(6), p. 41-44. Available at: http://appliedclinicaltrialsonline.findpharma.com/appliedclinicaltrials/IT+Articles/Solutions-Delivered-Not-Installed/ArticleStandard/Article/detail/334567?contextCategoryId=554 [accessed 28 April 2008].

Businessweek. 2006. *Software-as-a-Service Myths.* [Online]. Available at: http://www.businessweek.com/technology/content/apr2006/tc20060417_996365.htm [accessed 28 April 2008].

Carraro, G. & Chong, F. 2006a*. Architecture Strategies for Catching the Long Tail*. MSDN Architecture Center. [Online]. Available at: http://msdn.microsoft.com/en-us/architecture/aa479069.aspx [accessed 30 June 2008].

Carraro, G. & Chong, F. 2006b. *Software as a Service (SaaS): An Enterprise Perspective*. Microsoft Corporation. [Online]. Available at: http://msdn2.microsoft.com/en-us/library/aa905332.aspx [accessed 28 April 2008].

Chong, F., Carraro, G. & Wolter, R. 2006. *Multi-Tenant Data Architecture*. Microsoft Corporation. [Online]. Available at: http://msdn.microsoft.com/en-us/architecture/aa479086.aspx [accessed 30 June 2008].

CRM Landmark. 2007. *Is CRM SaaS or On-Premise Right For You?.* [Online]. Available at: http://www.crmlandmark.com/saasvonpremise.htm [accessed 16 June 2008].

Dagum, D. 2006. *An MSDN Architecture Chat About Software as a Service, with Gianpaolo Carraro.* Microsoft Corporation. [Online]. Available at: http://msdn.microsoft.com/en-us/architecture/aa479363.aspx [accessed 30 June 2008].

Erlanger, L. 2005. *A field guide to software as a service*. InfoWorld. [Online]. Available at: http://www.infoworld.com/article/05/04/18/16FEsasdirect_1.html [accessed 28 April 2008].

Finch, A. 2008. *The Rise of SaaS and Your Regulatory Risks.* E-Commerce Times. [Online]. Available at: http://www.ecommercetimes.com/story/61448.html?welcome=1213617636 [accessed 16 June 2008].

Fonseca, B. 2008. SaaS benefits starting to outweigh risks for some. *Computerworld.* [Online]. Available at: http://www.computerworld.com/action/article.do?command=viewarticlebasic&articleId=317340 [accessed 16 June 2008].

Georgia State University. 2008. *CIS 8020 - Systems Integration.*[Online]. Available at: http://www2.cis.gsu.edu/cis/program/syllabus/graduate/CIS8020.asp [accessed 18 October 2008].

Gruman, G. 2007a. *The truth about software as a service.* [Online]. Available at: http://www.cio.com/article/109706 [accessed 28 April 2008].

Gruman, G. 2007b. *Include Management Costs When Calculating Software as a Service Benefits.* [Online]. Available at: http://www.cio.com/article/print/111151 [accessed 28 April 2008].

IBM. 1998. *Improving Systems Availability.* IBM Global Services. [Online]. Available at: http://www.cs.cmu.edu/~priya/hawht.pdf [accessed 16 June 2008].

Knorr, E. 2006. *Software as a service: The next big thing.* Infoworld. [Online]. Available at: http://www.infoworld.com/article/06/03/20/76103_12FEsaas_2.html [accessed 28 April 2008].

Lashar, J.D. 2008. *The Hidden Cost of SaaS.* [Online]. Available at: http://www.destinationcrm.com/Articles/PrintArticle.aspx?ArticleID=48682 [accessed 16 June 2008].

Lauchlan, S. 2007. *Gartner predicts strong interest in SaaS.* [Online]. Available at: http://computerworld.co.nz/news.nsf/mgmt/DD87FF9423278B2ECC25733A0016CDE3 [accessed 3 July 2008].

Lawson, L. 2007. *The SOA, SaaS, Web 2.0 Connect.* [Online]. Available at: http://www.itbusinessedge.com/blogs/mia/?p=200 [accessed 16 June 2008].

Microsoft Corporation. 2008a. *Data Integration.* [Online]. Available at: http://msdn.microsoft.com/en-us/library/ms978572(printer).aspx [accessed 30 June 2008].

Microsoft Corporation. 2008b. *Encryption.* [Online]. Available at: http://www.microsoft.com/technet/prodtechnol/windows2000serv/reskit/distrib/dsch_key_tenv.mspx [accessed 18 October 2008].

National Information Standards Organization, 2004. *Understanding Metadata*. [Online]. Available at: http://www.niso.org/publications/press/understandingmetadata.pdf [accessed 18 October 2008].

Nieh, J., Yang, S.J. & Novik, N. 2000. *A Comparison of Thin-Client Computing Architectures*. [Online]. Available at: http://www.ncl.cs.columbia.edu/publications/cucs-022-00.pdf [accessed 18 October 2008].

Opengroup. 2005. *Introduction to Single Sign-On.* [Online]. Available at: http://www.opengroup.org/security/sso/sso_intro.htm [accessed 18 October 2008].

PC Magazine, 2008. *Definition of: downtime.* [Online]. Available at: http://www.pcmag.com/encyclopedia_term/0,2542,t=downtime&i=41950,00.asp [accessed 18 October 2008].

Service Level Agreement Zone. 2002. *The Service Level Agreement.* [Online]. Available at: http://www.sla-zone.co.uk/ [accessed 18 October 2008].

Schwartz, E. 2005. The True Value of SaaS. *InfoWorld*. November, 27(47), p10.

Schwartz, E. 2007. *Calculating the cost of SaaS.* [Online]. Available at: http://www.infoworld.com/archives/emailPrint.jsp?R=printThis&A=/article/07/03/20/13OPreality_1.html [accessed 16 June 2008].

Sharpy. 2008. *Definition of 'System'.* [Online]. Available at: http://www.sharpy.dircon.co.uk/index_files/DefinitionOfSystem.htm [accessed 18 October 2008].

Sieper, S. 2007. *Best Practices for SaaS.* [Online]. Available at: http://www.ebizq.net/topics/bi/features/8095.html?&pp=1 [accessed 23 June 2008].

Smoothspan. 2007. *Multitenancy Can Have a 16:1 Cost Advantage Over Single-Tenant.* [Online]. Available at: http://smoothspan.wordpress.com/2007/10/28/multitenancy-can-have-a-161-cost-advantage-over-single-tenant/ [accessed 16 June 2008].

Software & Information Industry Association. 2000a. *The Many Faces of Software Services.* [Online]. Available at: http://web.archive.org/web/20000815064749/www.trendsreport.net/software/2.html [accessed 16 June 2008].

Software & Information Industry Association. 2000b. *Software as a Service: Better, Cheaper, Faster.* [Online]. Available at: http://web.archive.org/web/20000815064749/www.trendsreport.net/software/3.html [accessed 16 June 2008].

Software & Information Industry Association. 2000c. *Software as a Service: Why We're Not There Yet.* [Online]. Available at: http://web.archive.org/web/20000815064749/www.trendsreport.net/software/4.html [accessed 16 June 2008].

Software & Information Industry Association. 2001. *Software as a Service: Strategic Backgrounder.* Software & Information Industry Association. [Online]. Available at: http://www.siia.net/estore/ssb-01.pdf [accessed 16 June 2008].

Sun Microsystems, 2008. *Applets.* [Online]. Available at: http://java.sun.com/applets/ [accessed 18 October 2008].

Sysmans, J. 2006. *Software-as-a-Service: A Comprehensive Look at the Total Cost of Ownership of Software Applications.* [Online]. Software and information industry association, 2006. Available at: http://www.siia.net/software/pubs/SAAS_TCO_WP.pdf [accessed 28 April 2008].

Tarzey, B. 2008. *The truth about software as a service.* [Online]. Available at: http://software.silicon.com/applications/0,39024882,39169781,00.htm [accessed 28 April 2008].

Wailgum, T. 2008. *Five Best Practices for Implementing SaaS CRM.* [Online]. Available at: http://www.cio.com/article/print/378313 [accessed 23 June 2008].

Wainewright, P. 2008. *Eight reasons SaaS will surge in 2008.* [Online]. Available at: http://blogs.zdnet.com/SAAS/?p=432 [accessed 28 April 2008].

Zucco, J. 2006. *Benefits of a software as a service model.* [Online]. Available at: http://searchnetworking.techtarget.com/generic/ 0,295582,sid7_gci1164670,00.html [accessed 28 April 2008].