



An Investigation into Web Services: A case of an Online Reservation System

A research study by Derick Wasonga Jabuto Odemba

Abstract

Distributed computing has been around for a while now. However, it is only after the explosion of the internet that the desire by businesses to integrate their business functions became really feasible. This study seeks to explore one of the modern technological implementations of distributed systems, the web services. The study will look at the motivation for the evolution of computer based information systems from data oriented to service oriented architectures. More specifically, it will investigate how web services have been used in the hospitality industry.

The research efforts of this study will culminate in the development of an online reservation system that can be used as a web service by other service providers in the hospitality industry. Considerable amount of time will be devoted to understanding the technologies used in building web services. However, identifying business benefits that firms can derive from implementing web services is also an important objective of the project. Finally, this study will be restricted to Simple Objects Access Protocol (SOAP) rather than Representational State Transfer (REST) based web services. The rationale for this is that SOAP is supported by well developed World Wide Consortium (W3C) standards besides having rich software libraries and toolkits. Other than that, the difference between the two web service technologies is very small.

Key words: Distributed systems, Web Services, Web Services Description Language (WSDL), Simple Objects Access Protocol (SOAP), Universal Description, Discovery and Integration (UDDI).

Acknowledgements

In the course of this study, I got assistance from a number of people, a few of whom I wish to mention. Many thanks go to Daniel Kinyanjui and Teresia Kochumba, librarians at Oshwal College. They were as helpful to me as they could possibly be. A debt of gratitude is due to Dr. Lawrence Nderu whose guidance, scrutiny and critique of my work was invaluable during this study. To my family, the Odemba family, I am humbled by the unwavering financial support that I received from you during the entire period of this project. Thank you. Finally, to those who helped me in one way or the other, but whose names I have not mentioned, know that your assistance has not gone unappreciated.

Chapter One

Introduction

The number of people accessing and using the internet around the globe has dramatically increased in recent years. This unabated penetration of the internet is forcing businesses to take their services online in order to reach out to a wider market. This calls for an implementation of a technological infrastructure that can allow interoperability between different systems or applications in an efficient and seamless manner. Web services provide the solution to this business need. They are developed in a way that allows applications to communicate or talk to each other irrespective of their location, IT infrastructure on which they run or programs used to develop them. A web service has an abstraction layer in its architecture that separates the technology from the applications or services to make it easy for applications to interact over the web without interference of the technological infrastructure. This is actually made possible by the use of standard technologies such as XML, SOAP, WSDL, UDDI, ebXML, HTTP and others that are at the core of the architecture of web services.

Central to the operations of web services are ontologies. Ontology enables systems, people or organisations to have a shared understanding of terms or vocabulary used within a particular domain. Ontologies were first used in artificial intelligence but are now being widely used in various fields (Noy and McGuinness, 2000). They explicitly specify intended meaning of terminologies used in a given expert domain. The main goal of ontologies is to enable interaction between parties in a system. In the context of web services, ontologies construct semantics used by web services to enable applications to interoperate easily.

While the uptake of web services technology has not been as fast as many industry proponents and supporters would have wanted, there is enough evidence that more businesses will adopt web services in the future. With a few success stories of web services implementation and deployment, and so much effort being put on web services security standards by major corporate technology vendors and industry standard organisations, this future is not far off.

The ultimate goal of this project is to develop an online reservation system that will be exposed to hospitality industry service providers as a web service. However, this paper in addition, tries to explain the technical composition of web services and the technologies and standards used to implement them. It also discusses the various techniques used to address

the main challenge of security that web services need to overcome in order to attract widespread use among businesses. Finally, the paper outlines some of the advantages of using web services by businesses and the future of the technology in general, and of its application in the hospitality business in particular. Two examples of companies in the hospitality business that adopted web services have been used to demonstrate how useful web services can be to the hospitality sector.

This document is made up of seven chapters, each of which is dedicated to a specific aspect of the project. Other than the introduction, the remaining six chapters are covered in chronological order in the following sections.

Literature Review: The literature review forms the second chapter of the document. It examines web services in details to enable readers gain an understanding of what they are by investigating technologies used to implement them and how they work. It further gives two cases of companies in the hospitality business that have implemented a web service and how it has impacted their businesses.

System Development Methodologies: This is the third chapter of the document and covers system development methodologies. The chapter begins with a general overview of methodologies and discusses their development over time to date. It further looks at selected methodologies and examines their strengths and weaknesses. The choice of the methodology used in this project and the justification for selecting it is also discussed in this chapter.

Requirement Analysis: The fourth chapter, requirements analysis outlines the various techniques that were used in gathering, analysing and validating the requirements for the system. It goes further to give specifications of those requirements in a way that can be translated into a design. Requirements analysis includes examining both functional and non-functional requirements of the system.

System Design: System design is the fifth chapter of this document and is concerned with the process and techniques used in converting the requirements specification into a model that can be used to write the codes for the system. The design process falls into three categories of conceptual, logical and physical design. The deliverable of this chapter is a model that forms the basis of developing the various components as well as integrating them into a single functional system.

Implementation: This second last chapter of the project discusses the programming languages or software used to build the application. It also gives details of how codes of the various units and of the entire system are written. The chapter also describes how the system works as one whole.

Evaluation: Evaluation is the final chapter of this document and describes the tests that the system has been subjected to and how they have been conducted. Typically, it gives details of the test data used for the unit, integration and system testing.

Chapter Two

Literature Review

2.0 Introduction

This study aims to present web services to readers in comprehensive but simple terms. It will examine the fundamental components of a web service, what web services are as well as their application in the hospitality industry. To aid in this, an in-depth analysis of two cases of web service application in the hospitality industry will be carried out. This exercise will also find out the technological challenges that need to be overcome for web services to gain widespread adoption by businesses. Finally, it will give an insight into the direction future developments in the use of web services in the industry is likely to take.

2.1 Distributed Systems

To understand distributed systems, one first needs to appreciate distributed computing, the underlying technology upon which distributed systems are built. There are varied definitions of distributed computing, but authorities on the subject agree that distributed computing involves the study of how computers are connected to each other using networking technologies. The purpose of connecting computer systems together is to allow computing resources to be shared across the organisation (Natalia and Olifer, 2005).

In a distributed system, applications are divided into smaller chunks which are consequently processed on several computers on a network. These computers can be clients or servers depending on the network infrastructure implemented by the organisation concerned (Goldman, 1997). Software Oriented Architecture enables businesses to implement a more advanced, dynamic and a service oriented version of distributed system in the form of web services. The Internet and Automatic Teller Machines (ATMs) are some typical examples of distributed systems. Figure 2.1 demonstrates a simple distributed system for selling books online.

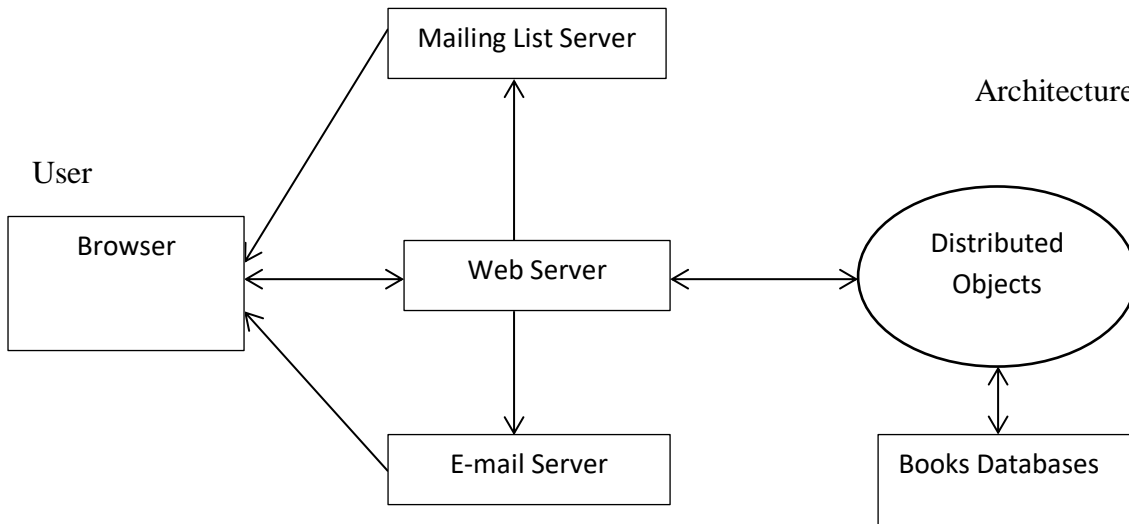


Figure 2.1: Example of a distributed system (Ince, 2004)

Figure 2.1 depicts an online book store that users can buy books from online. Users are able to browse through the catalogue before choosing and purchasing the book they want. They interact with the books database via the browser connected to the web server. Once the book store sends the ordered books to the buyer, he or she receives an email notification from the store. Books databases store the details of the books. The distributed objects enable the book databases to communicate with web server, which in turn is connected to the E-mail server and the mailing list.

2.2 Service Oriented Architecture

A study of web services necessitates an understanding of the concept of Software Oriented Architecture (SOA). This is because, though there are other technologies, web services provide one of the best approaches to implementing an SOA. SOA has no universally agreed standard definition. Where there is no dispute though, is the fact that SOA does not describe a product or a service. It is an infrastructure, a structure, a design style that enables heterogeneous Information Technology (IT) systems or applications to share each other's functionalities. *Service Oriented Architecture (SOA) is a business-centric IT architectural approach that supports integrating your business as linked, repeatable business tasks, or services* (Lin et al., 2012).

2.3 Web Services

A web service is an application that resides and runs on the web, and provides services to other applications over Hypertext Transfer Protocol (HTTP). In essence, a web service is a distributed application, which in basic terms comprises a service and a client. In SOA terms, a client is also referred to as a consumer or requestor (Kalin, 2013). Web services can be implemented within the same organisation or between two business organisations trading with each other. In this project, the web service will provide an inter-enterprise service. What sets web services apart from other distributed systems is its ability to run or operate independent of the underlying IT infrastructure. In other words, a web service provides its services to other applications without paying any attention to the hardware and operating system on which those applications run or the programming language in which they are written. The use of open standards and protocols is responsible for this much needed feature of web services.

There are several protocols that have been developed to enable interoperability between applications. HTTP, XML, SOAP, WSDL and UDDI are some of the protocols that a web service requires to function. However, other additional standards that address web service management and web service security have also been developed. The succeeding paragraphs ignore HTTP and delve into the remaining four fundamental protocols.

2.3.0 Extensible Markup Language (XML)

XML is arguably the most important of the standards and protocols used in developing web services. It is a language that developers use to define other languages, particularly markup languages such as HTML. XML documents are in text format and are both human and machine readable. The documents are presented in a standard data format that makes them easily exchangeable between applications running on very different computing platforms. Typically, the exchange of data takes place between consumers and providers of web services (Endrei *et al.*, 2004). As will be seen shortly, WSDL, SOAP and UDDI are coded in XML tags.

The development of XML Schema by World Wide Web Consortium (W3C) to replace Document Type Definition (DTD) as a standard for defining XML languages has made XML popular in developing web services. DTD's capability was limited. For instance, it did not accommodate date and number data types among others.

2.3.1 Web Service Description Language

WSDL is an XML based standard that is crucial to the implementation of web services. As the name suggests, WSDL's role is to describe the web service so that consumers (other services or applications) can know what services are offered by the web service, the location of the services on the web and how to access them. The description includes information such as the services offered and their format, Universal Resource Identifier (URI) of the service and the address of the web server that hosts the service. This information is stored in the web service registry, the UDDI. Consequently, applications that wish to use the service must discover a WSDL document in the UDDI (Ort, 2005).

A WSDL document is made up of four elements, each of which plays a specific role. The elements are types, message, portType and binding. Types specifies the XML Schema or data types that the web service uses. Message contains the data elements for each operation. portType outlines the operations that can be performed as well as the messages involved in the operation. Each portType has a protocol and data format clearly defined by binding. Binding enables WSDL information to be transported by SOAP. It is therefore safe to say that WSDL defines the web service interface.

2.3.2 Simple Object Access Protocol (SOAP)

SOAP can be described as a messaging or communication protocol. It is used to transmit XML data between a web service provider and a consumer over the internet. SOAP employs the services of HTTP to enable the applications exchange data between them. Due to security threats posed by operating online, many implementations of web services now use Hypertext Transfer Protocol Secure (HTTPS). Any communications over HTTPS are encrypted to make them secure. SOAP is not affected by any changes in the operating system, network configuration, hardware or the programming languages in which the participating applications are written as it is independent of them. This, alongside support from major industry vendors and its ability to self-correct has made SOAP the messaging protocol of choice for developers implementing web services.

A SOAP message is made up of an envelope. An envelope is nothing more than a set of XML tags written to abide by rules outlined in the SOAP specification. All the information required to accomplish communication between the sender and the recipient, including the sender's specific request and how the message is to be interpreted is included in the envelope

(Englander, 2002). An envelope is the main element in SOAP message and has two sub elements, an optional header and a mandatory body. If present, the header will contain supplementary information to aid message exchange between two applications. The body on the other hand consists of the actual message being sent from the sending to the receiving application.

2.3.3 UDDI

In the implementation of a web service, UDDI acts as a registry of services that a web service offers. It benefits from the services of other standards such as WSDL, SOAP and others to publish a web service and enable applications (web service clients) to discover and get access to services in a web service. UDDI does this by storing web service interfaces as described by WSDL (Newcomer, 2002). UDDI functions in a similar way to the Internet's Domain Name Service (DNS). Specifically, a client wishing to consume a web service finds its Universal Resource Identifier (URI) in the UDDI registry and subsequently sends a request to the web service using an XML based message over the internet. On receipt of the message, the web service responds by sending a reply to the client in the same format as the request it received.

It is important to take cognisance of the fact that the exchange of data between the web service and the client is facilitated by SOAP. To be precise, SOAP provides a framework for transporting XML data over the internet. A more detailed explanation of how SOAP, WSDL, UDDI and other associated technologies work together to deliver a functioning web service is covered in the next section, web services architecture.

2.4 Web Service Architecture

Web Services Architecture (WSA) describes the physical layout of the various components of a web service and how those components work together to deliver services to web service clients. Since applications that interact with each other in a web service are varied and run on diverse IT infrastructure, the WSA includes additional technologies and specifications that deal with security, management of the flow of processes between web services and control of communication through many networks and between web services. Put in another way, WSA provides a framework and a basis for understanding how web services work.

Most writers on the subject identify two ways in which WSA can be looked at: by looking at the roles of each component and examining the web services protocol stack. In this literature review, both approaches will be covered since each offers a unique perspective to the topic.

The former sees web services as a combination of three basic interactive roles, namely Service Provider, Service Registry and Service Requestor. Working with find, publish and bind operations, the service provider, service registry and service requestor are able to deliver a working web service for applications that need to consume their services.

Simply put, a service provider is the owner of the business function that is of interest to other applications. Technology wise, a service provider describes the infrastructure on which the service runs or on which it can be accessed by clients. Using WSDL, the service provider describes the service and publishes it in the service registry. That way, the service requestor is able to discover the service and send a request to the provider. The service requestor is the client. It is usually the business that needs to use the web service to solve a specific problem.

In technical terms, a service requestor is the application that requires the services of a web service. The requestor usually finds the details of the service in the registry and uses that information to initiate communication with service provider (Dustdar and Schreiner, 2005). A web service cannot function without the service registry, which acts as a directory of all services offered by the service providers. Here, details such as contact of the service provider (the company offering the service), location of the service on the internet and other technical information that the requestor may need to use the service is stored. A diagrammatic illustration of the role based approach of WSA is shown in figure 2.2.

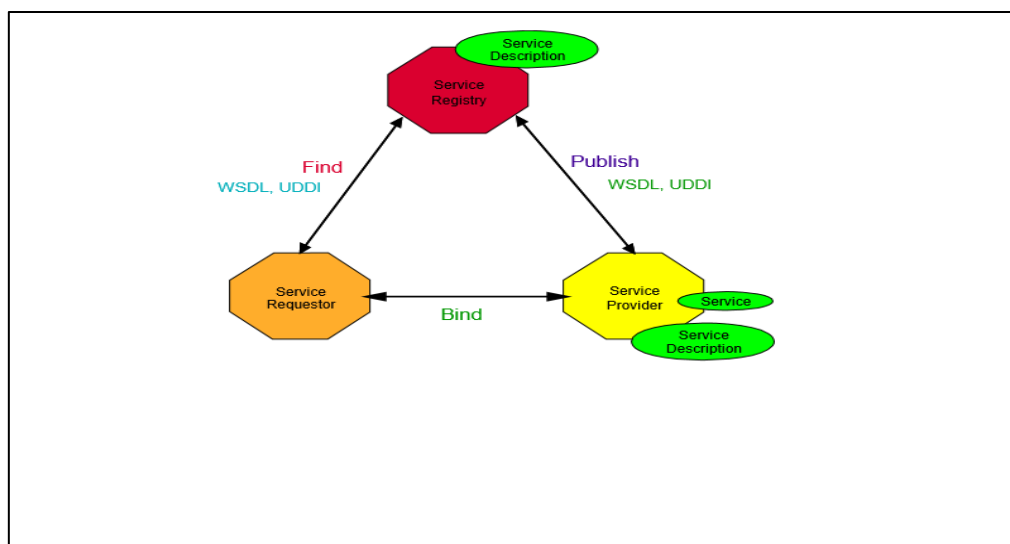


Figure 2.2: Web Services Architecture

(http://users.cs.uoi.gr/~pitoura/courses/ds04_gr/webt.pdf)

Web services stack presents a web service stack as a layer of functions and requisite protocols or standards used in each layer to enable a web service deliver services. There exist as many variations of the web services stack as there are corporate organisations, vendors or institutions supporting web services technology. Nonetheless, the fundamental technologies and components of the web services stack remain largely similar in the various versions. In this review, the IBM variant has been chosen to illustrate the web services stack as shown in the diagram on the next page.

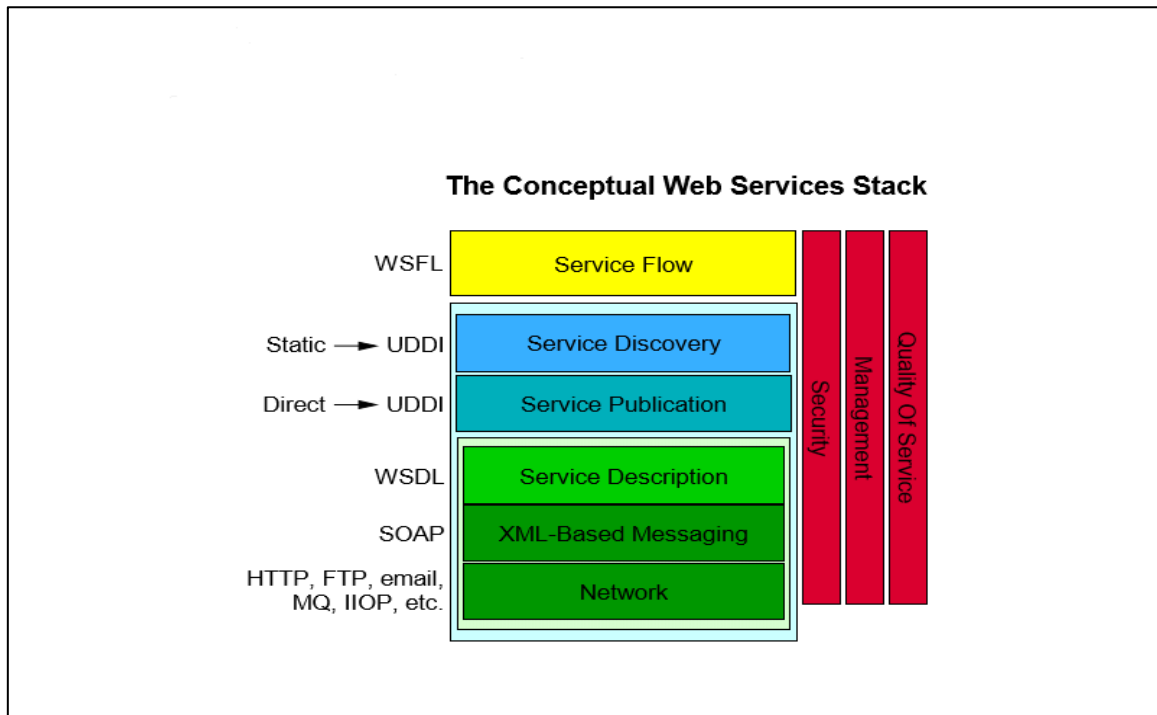


Figure 2.3: Web Services Stack (http://users.cs.uoi.gr/~pitoura/courses/ds04_gr/webt.pdf)

The stack is made up of six layers which can further be consolidated and classified into three. Each layer has a corresponding protocol that enables it to perform the functions expected of it. Quality of Service, Management and Security are performance metrics that are required to be dealt with at each layer of the stack in order to implement a functional web service. The upper layers are functionally dependent on the layers below them (Kreger, 2001). The network layer is the most critical in the web services stack as it implements the internet on which web service resides. The internet protocols such as HTTP, File Transfer Protocol (FTP), Simple Messaging Transfer Protocol (SMTP) and so on operate at the network layer.

Following the network is the XML- Based Messaging layer which is basically the communication layer. SOAP is the preferred protocol used at this level due to its simplicity,

support of XML documents and the three basic web service architecture operations of publish, find and bind. The Description layer is responsible for defining a web service and is managed by WSDL as had been earlier explained under WSDL section. The three lower layers of the web services stack are sufficient to implement a web service that can be accessed and consumed by other businesses. However, building an intra-enterprise web service may require additional technologies. The focus of this study will be to develop an inter-enterprise web service in the form of a reservation system.

2.5 Benefits and Challenges of adopting Web Services

Web services offer numerous benefits to organisations as well as technical people like developers of applications. On top of the list of value added services that a web service offers is interoperability. Web services enable applications developed in different programming languages and running on completely different platforms to interact seamlessly with each other. Its reliability on open standards and protocols makes it natural for web services to deliver this feature. Many advantages that organisations and developers enjoy from using web services stem from its ability to integrate applications, which in the case of a business are specific functions.

Web services can help businesses to considerably reduce IT expenditure. Since it operates on the web, businesses can be able to expose their services without incurring new expenses on hardware or IT infrastructure. Moreover, web services allow applications to be reused thereby making developing new software solutions easier, faster and cheaper. The loosely coupled nature of web services also makes it easy for businesses to scale up a business function or solution without making any significant changes to their IT infrastructure (Endrei et al, 2004). Changes to the IT infrastructure (that is, changing operating system, network infrastructure or a web server) will not interfere with how the web service works. As a result, there would be no need to make any changes in the application exposed as a web service.

Managing to keep IT solutions current to match the pace of rapid changes in the business needs as dictated by fast evolving business environment is a challenge to many organisations. Web services help businesses to deal with this problem without compromising quality of service delivery. By its very nature, a web service interface tends to remain the same. Therefore, changes to future versions of an application have very minimal or no disruptive effect at all on the service, meaning that consumers can continue to be served despite changes in the web service's source code.

Another major challenge that businesses intending to implement web services have to contend with is that of security. In fact, one of the quality metrics of a web service is its ability to provide services to its clients securely (Du, 2004). An ideal web service implementation needs to meet minimum security requirements such as proof of identification by service requestors, authentication, integrity and confidentiality of data being transmitted, ability to perform an audit trail of all transactions between applications as well as provide evidence that the information sent and received are identical (Endrei et al, 2004). More often than not, some of these requirements are not met by web services currently being implemented because of the absence of standards or inability of developers to find well established technologies to build applications.

This study is not intended to investigate web security as a topic since it is a whole subject in itself, and one on which a lot of work is still being done. Nonetheless, it would be unthoughtful to fail to explain how web services can be implemented securely. Besides the inbuilt security features implemented at each layer of the web services stack, there are a number of specific techniques that are being used to address the security concerns arising from deploying web services. Some of these techniques are discussed in the succeeding paragraphs.

HTTP Authentication: This is the simplest security technique implemented in networks. In this security implementation, the service requestors seek authorisation to access the service provider's server by identifying themselves. This is often accomplished by entering a username and a password and in some cases biometric access.

Secure Socket Layer (SSL): SSL is a combination of SSL record and SSL handshake protocols that ensures data is transmitted securely between two applications on the web. Data from the sending application is encrypted and transmitted over HTTPS protocol to the receiving application. Upon receipt, the data gets decrypted by SSL. This process solves privacy and confidentiality, and authentication and integrity problems. Because of these capabilities, SSL has become a popular security implementation in online business transactions (Du, 2004).

Web Service Security (WS-Security): WS-Security is a security specification that was originally developed jointly by IBM and Microsoft under the Organization for the Advancement of Structured Information Standards (OASIS). It specifically deals with SOAP message security in web services. WS-Security makes use of XML signature and XML encryption to protect XML

formatted SOAP messages. XML encryption was developed by W3C and implements encryption selectively. In other words, it only encrypts parts of an XML document making the SOAP message secure during data exchange (Toms, 2004). An additional benefit of XML encryption is that it secures the SOAP message even when it is not being transmitted, thereby guaranteeing the confidentiality of the message. Also developed by W3C, XML signature aims to enable web services maintain the integrity of a SOAP message during transit by signing the most important part of a SOAP message. XML signature employs the services of other hashing algorithms to find out if a message has been altered or interfered with during transfer.

2.6 Web Services in the Hospitality Industry

Most businesses in the hospitality industry turn to web services in order to reduce costs and earn more profits. The ability of web services to bundle several services together into a single interface within the business and its B2B offering deliver this principal benefit to businesses. First and foremost, it is necessary to state that this literature review does not focus on the use of web services in a particular country, region or continent. Rather it is concerned with how the technology can be used in the hospitality industry. Though other sources have been used to provide background information on the use of web services in hospitality business, this literature review has used two companies, e-Travel (www.e-Travel.gr) and Vail Resorts (www.vailresorts.com) as case studies.

As has been indicated earlier in this project, web services can integrate various functions within a business or between two or more businesses. To develop intra-enterprise applications, XML schema and SOAP are sufficient. The reason for this is that there is no need for an interface where a description of services is stored since the various applications will already be sharing network addresses, name and data definitions (Peters, 2002). Members of the software development team and users will share such information as well. In essence, such a web service does not require WSDL and UDDI to be implemented. According to Peters (2002), intra-enterprise web services can be implemented in two ways. One is to modify the existing application to transform it into a web service. Many businesses often shun this approach due to high costs involved in effecting the change. The second and the more popular method is to have an Enterprise Application Integration (EAI) adapter placed in front of the candidate application to make it appear as a web service.

In addition to WSDL and UDDI, implementing inter-enterprise web services for hospitality businesses may require an additional standard known as electronic business XML (ebXML)

to achieve seamless and secure interoperability. Like the other web services technologies, ebXML is platform agnostic. It also depends on the known internet protocols such as Transmission Control Protocol/Internet Protocol (TCP/IP), HTTP, XML, SMTP, FTP and others to do its work. The development of ebXML was initiated and supported by OASIS and United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT) to enable businesses of all sizes irrespective of where they are around the world to easily conduct business electronically with each other by exchanging XML data (Kotok and Webber, 2002).

To be precise, ebXML automates B2B interactions. Its architectural design incorporates both business and functional specifications. The business specification deals with issues like semantics of data used in business transactions, business collaboration agreements and processes. The functional specification on the other hand addresses the technical aspects of the implementation. Security of data transfer, discovery of the service and configuration of the service interface are some of the aspects dealt with by the functional specification.

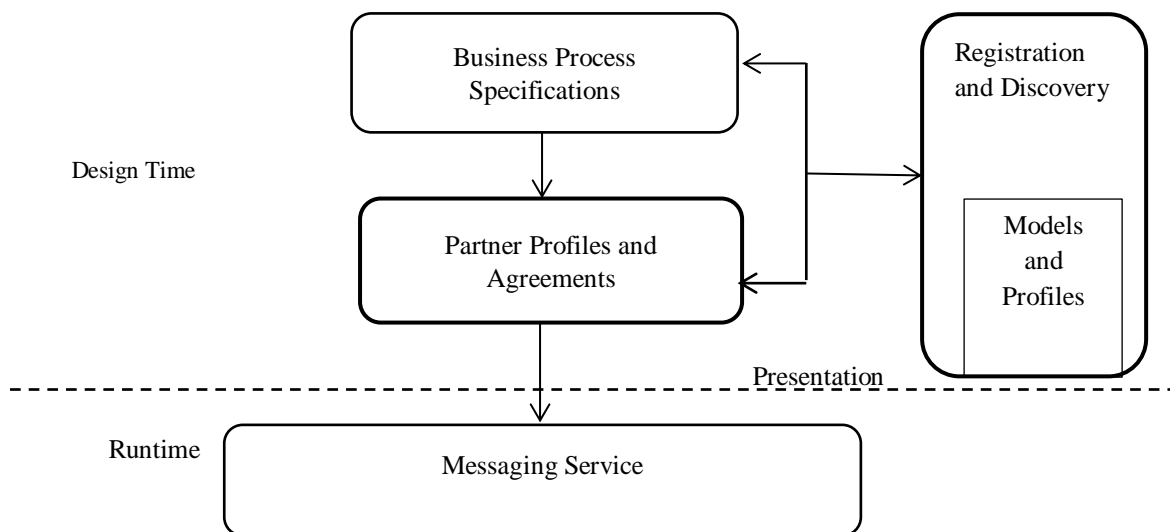


Figure 2.4: ebXML Architecture (https://www.tutorialspoint.com/ebxml/ebxml_tutorial.pdf)

Given that the online reservation system to be developed will be exposed to different hospitality industry service providers, ebXML alongside other technologies will be used in its implementation.

2.6.1 e-Travel

e-Travel provides a range of services to passengers both domestically and internationally using a web service. Travellers are able to buy air tickets, rent cars and buy travel insurance using their service. Their greatest challenge was to get an IT service that could effectively support their core strategic business goals of reducing the time to market their products and services, business expansion beyond Greece and reliable delivery of quality services to their customers. Their initial IT focus was on providing a reliable IT infrastructure, in which case, IT staff spent their time on housekeeping tasks. This was in complete variance with what it needed to do in order to improve customer service. The company decided to implement a web service and chose Amazon Web Services (AWS) as their host provider.

The migration took place in 2012 and was relatively simple with only a few hours down time. Effectively, what e-Travel does is just develop applications which offer its various services and expose them to the web for customers using AWS. AWS has tools that e-Travel developers use to develop applications. It has a booking and ticketing system, and 11 websites offering various services all hosted by Amazon Virtual Private Cloud. The results of moving to web services have been impressive. e-Travel's experiences reduced down times as AWS guarantees up to 99.9% system availability. Consequently, customers have access to services consistently. This has boosted their confidence in the company's service delivery. Expansion of their business portfolio services has also been fairly easy, fast and less costly since the company does not need to introduce new hardware such as servers and related accessories every time it introduces a new service in response to the changing customer needs. Implementing the web service enabled the company to increase its global sales by 30%, making it possible for it to achieve its goal of global business expansion.

Another key value addition of the web service to e-Travel is that it now takes much shorter to market new services, and to do that in a more effective way. This has given it an edge in this very competitive industry. e-Travel now considers AWS as an essential and inseparable part of its business operations and one that is key to its strategic growth. AWS makes the work of e-Travel developers extremely easy as it provides a managed service. Developers have access to prototypes and ready codes that only require little modification to meet their requirements. It is important to note that many service providers do not offer this kind of service. The typical scenario is that businesses develop their own applications and only need a registry to

publish their services. This project will take the latter approach to implementing web services while developing the online reservation system.

2.6.2 Vail Resorts

Vail Resorts is a hotel group based in the United States of America. It has several luxurious resorts in Colorado and California. Vail Resorts has about 15000 employees as well as temporary or contract staff. Their IT infrastructure included IBM i Report Program Generator (RPG) applications for running their internal business functions and a Windows web application for customer self-service. But the system did not meet the business requirements of Vail. The server and the application system were not integrated. As a result, the staff had to input the same data twice. This did not only waste valuable time, but also resulted in errors in the booking of rooms. Overbooking was particularly common during peak seasons as the IBM i system could not be updated fast enough.

An additional challenge that Vail faced was that of training new staff to use their system. Given the nature of the system, new staff did not learn the system as fast as the management wanted. This definitely affected the productivity of staff and compromised service delivery to customers. Vail was in need of a system that would help them do away with the double entry of data between their two systems and the problems associated with it. However, they were also keen to maintain the integrity of data held in the RPG application as they continued to rely on its business rules and logic. They found out that web services would be the appropriate technology solution to this problem.

Vail chose Looksoftware, a software company that uses IBM i platform to implement web services, RPG Open Access and integration among other services as their provider. Looksoftware provided an SOA that enabled the integration to take place smoothly without any need to change the RPG code for the business logic. The user interfaces, mostly used by the reception and reservation staff were however modified as a result of the integration. This certainly was a straightforward task for developers.

Vail's integration of the two systems using web service technology delivered tangible benefits. First, staff spent less time keying in data because implementing web service eliminated the need to key in the same data twice. This improved staff productivity. Second, there were fewer errors in the data entered into the system. Third, staff training also took a

shorter time after implementing the web service since the integration resulted in only a single interface. Last but not least, the integration led to improved customer satisfaction.

2.6.3 Comparison of the scenarios

While both companies had problems that required them to implement web service as the appropriate solution, their IT infrastructures were slightly different. And this had an influence on the type of web service that they implemented. e-Travel's implementation is a B2B web service as it allows its services to connect to other industry service providers such as car hire companies, airlines and insurance companies. It used AWS web service for its back-end, front-end applications as well as running the websites for its services. Vail Resorts on the other hand, used the web service to integrate the intra-enterprise functions. Its system is only being used within the enterprise network to allow easy management of reservation and customer management. Customers have easy access to their services by accessing them through its (Vail's) websites, making it a B2C web service implementation.

A common feature of the integration is that the transition was smooth in both cases. In either case, the underlying code for business logic was not affected during the transition. The business benefits are similar as the companies each concentrated on packaging and marketing the products and services rather than worrying about the technology infrastructure. By implementing web service, both companies experienced improved development staff productivity, better service delivery leading to higher levels of customer satisfaction and reduced costs. Both companies opted for a well-established service provider as reliability and continued availability of services were major motivations for implementing the technology. This is understandable given the size of the client base for both companies. However, this project will involve developing a much simpler web service application.

2.7 Conclusion

This study has confirmed an earlier statement that SOAP is a more popular implementation of web services than REST despite the fact that a REST web service is easier to implement. And like any new technological innovations, the introduction of web services was marked by enormous hype regarding the new business and technological benefits that it would offer. However, the low pace of the uptake of the technology by businesses has demonstrated that web services are still largely futuristic. Most businesses that have implemented web services have confined it to sharing various functions within the enterprise.

Business to Business (B2B) implementations of web services are still comparatively fewer as many businesses are still not confident enough to take their services beyond the firewall. But with major technology companies such as Microsoft, BEA, Oracle, IBM and standards organisations like W3C, OASIS, Web Services Interoperability Organization (WS-I) and others supporting the technology through the development of security standards, many more corporate organisations particularly in the hospitality industry will deploy inter-enterprise web services in the not too distant future. The use of ebXML as a standard in building web services promises to make this a reality not only for big businesses, but also for small ones whose lack of financial capability is another deterrent to implementing inter-enterprise web services other than security. Moreover, many businesses in the hospitality industry acknowledge that implementing B2B web services can greatly add value to the services they offer.

Chapter Three

System Development Methodologies

3.0 Introduction

Software development is a laborious and expensive exercise. It is therefore important that the undertaking is carried out in a manner that not only delivers an application that meets its functional and non-functional requirements, but also makes the work of all the project stakeholders bearable during the development process. This cannot be accomplished without following a well-defined, and in many cases a clearly documented set of guidelines and procedures outlining what activities and tasks are to be completed and the order in which they are to be completed. Also important are any tools and techniques used by software developers to carry out the activities and perform the tasks involved in developing software. In a nutshell, this is what a system development methodology is all about.

The development of the methodologies was necessitated by the software crisis of the 1960s. Frequent and costly failures of software projects forced industry practitioners to develop mechanisms that would reduce the rate and frequency of failures of software projects. Computer scientists and industry practitioners realised that software development was not only a technical undertaking. The people and organisational aspects of developing software were nearly as important as the technical ones. This led to the development of formal mechanisms for developing software that incorporated both people and technical aspects of software development. The term methodology was coined to refer to these mechanisms. These early system development methodologies were linear and stepwise (Simão, 2011). A number of methodologies including Waterfall, Spiral, Rational Unified Process (RUP) and V-model belong to this class of system development methodologies.

Even though these methodologies were continuously refined in order to improve the productivity of software developers and software quality, it soon became apparent that more flexible and user centric approaches to software development were needed. This is because these traditional methodologies emphasised technical specifications and process documentation but paid lip service to user involvement in the process. As a response to these deficiencies of the traditional methodologies, a new set of modern methodologies that gave more attention to user needs and allowed iterative and incremental development approach were developed. Rapid Application Development (RAD) was the first among these new set of

methodologies to be developed in 1991. The development of RAD has inspired the formulation of even more advanced methodologies that are highly responsive to the development requirements of software, users and software developers. These methodologies are generally referred to as Agile Development Methodologies and have been in use since 2001.

Each of the many methodologies has been developed by different individuals or organisations and is therefore based on different philosophies that provide some insight on understanding them (Avison and Fitzgerald, 2008). A philosophy outlines the principles and values of the methodology and its developers. Some methodologies are more detailed, sophisticated, and difficult to use. In practice, a combination of different methodologies is always employed in developing software (Despa, 2014). In this project, RAD will be used as the main methodology. However, elements of Waterfall and Rational Unified Process will be incorporated in the process. The following paragraphs discuss a few software methodologies that represent both the traditional and the modern software development approaches.

3.1 Waterfall

The waterfall system methodology is one of the most popular System Development Life Cycle (SDLC) approaches to developing information systems. It is a linear sequential methodology in which one phase of software development process must be completed before proceeding to the next. Each of the phases of the process must have a deliverable and must also be clearly documented. This is actually the greatest advantage of the waterfall model. Its major drawback is the breaking down of the software development process into phases which must be completed in a sequential fashion. This conflicts with the actual reality of software development where development activities and processes often overlap. This methodology would give good results in small software projects with clear requirements. Used alone, the waterfall methodology can lead to development of applications that fall short of the user requirements. According to Sommerville (2004), the waterfall methodology is made up of five distinct phases. The five phases of waterfall are as shown in the figure 2.5.

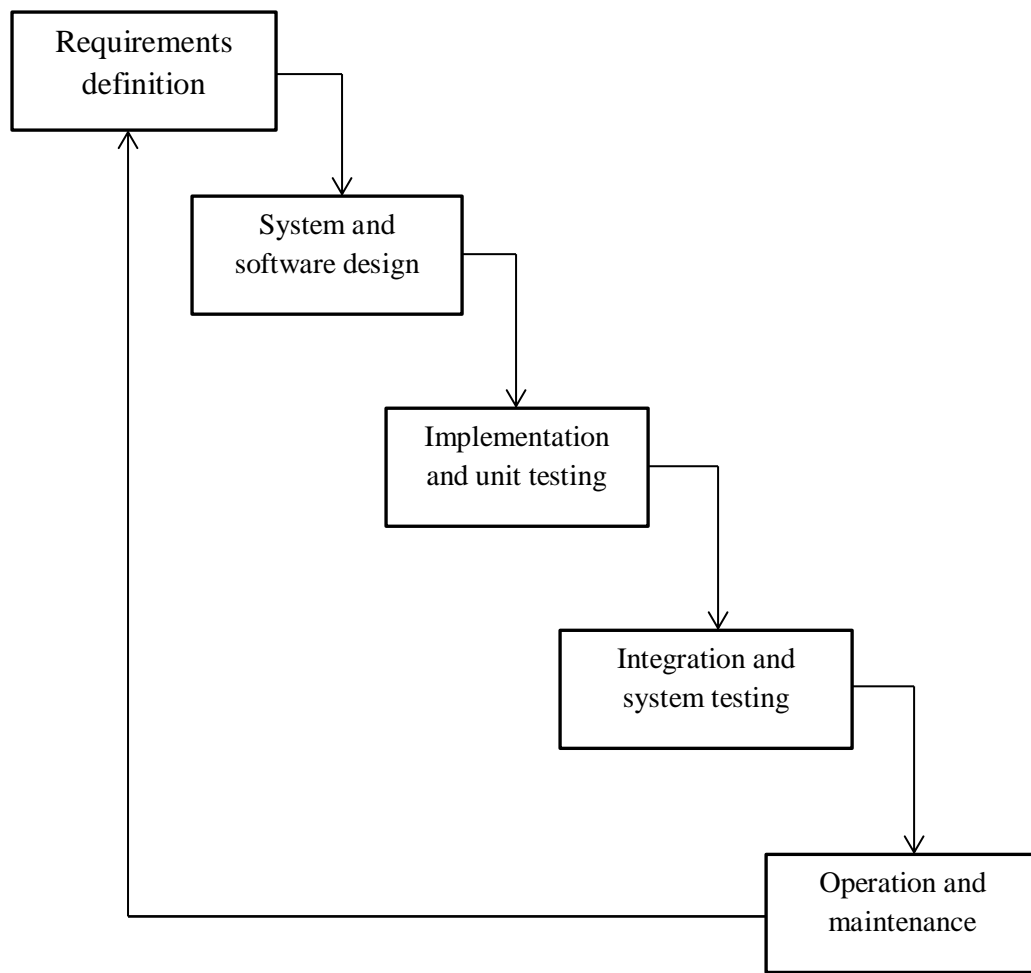


Figure 3.1: The software life cycle (Sommerville, 2004)

Each of the stages in the waterfall methodology is discussed in the following paragraphs:

Requirements analysis and definition: At this stage the analyst or developer confers with the system users to agree on and set out the scope, objectives and requirements of the proposed system including resources that will be needed to accomplish the project. These are clearly documented to be used as a reference and a guide throughout the life of the project.

System and software design: The system involves defining and specifying the overall system infrastructural requirements that the proposed system will run on in terms of hardware, software and networks. Software design on the other hand deals with defining the system functionalities and how the various modules or components of the proposed system interact with each other to achieve the overall functionality of the software.

Implementation and unit testing: The various modules of the system are coded as per the design specifications of the system at the software design stage. Each of the individual modules is tested to ascertain that they are free of bugs and are doing what they are intended

to do. This is normally one of the most tedious phases of the software development cycle particularly if the design specification is not done well.

Integration and system testing: The goal of this stage is to ensure that the various components of the system are able to work together as a unit. The system can only be delivered to the customer if the integration and system testing is successful.

Operation and maintenance: This is the final level of the waterfall methodology where the developed system is installed and used by the end users. Any errors that the developers were not able to identify during the early stages of the system development are corrected at this stage.

3.2 Rational Unified Process

The efforts that eventually culminated in Rational Unified Process (RUP) methodology began back in 1995 with work on Rational Approach and Objectory process 3.8 by Rational Software. The work led to the development of Rational Objectory Process 4.0 in 1996. RUP was finally developed in 1998 as the successor to the Rational Objectory Process 4.1, itself released in 1997. But since the takeover of Rational Software by IBM in 2002, RUP has been identified as an IBM process. RUP uses Unified Modelling Language (UML) technique which relies on the use of use cases as the basis of system design, implementation and testing (Avison and Fitzgerald, 2008). It is thus accurate to say that unlike the waterfall model that puts more emphasis on the technical details of the development process, the RUP model focuses more on the business aspects of the software development process (Sommerville, 2004).

It is a phased model that incorporates iteration and incremental software development approach to building applications. The four phases that make up RUP are Inception, Elaboration, Construction and Transition. Intertwined with the four phases are nine activities that are carried out during the development process. In RUP terminology, these activities are referred to as workflows. Figure 2.6 illustrates the relationship between the phases and the workflows in RUP.

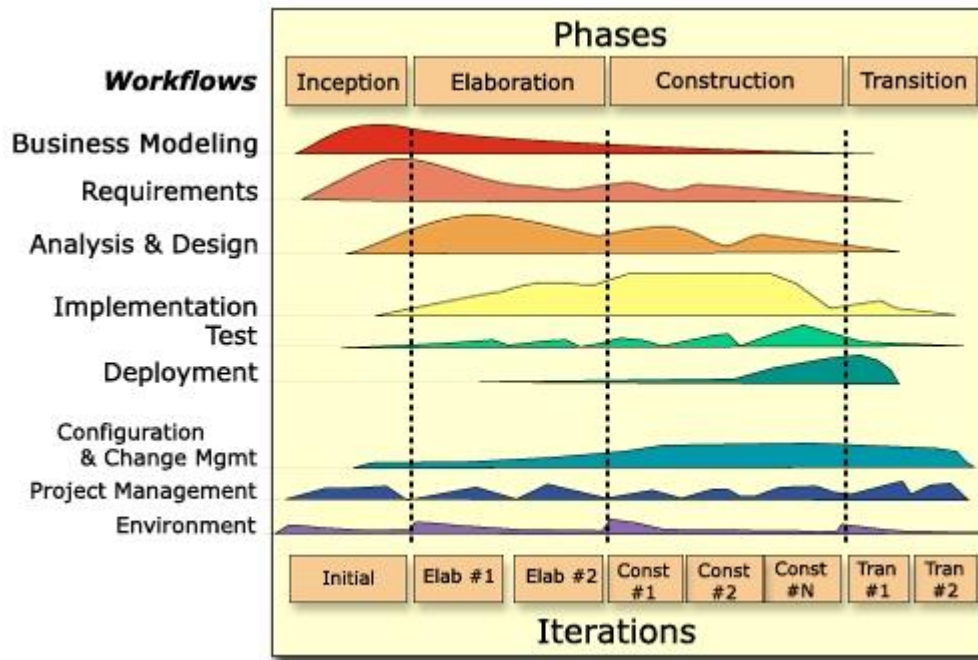


Figure 3.2: Phases in the RUP (https://era.nih.gov/docs/rup_fundamentals.htm)

Inception: The inception stage indicated as initial in figure 2.6 focuses on identifying a solution (the system idea) for the business problem to be solved. The resource requirements of the system are established at this level. The analyst or the lead project manager should put up a strong business case that shows clearly how the system will impact business. The project can be endorsed or rejected at this level depending on the outcome of this initial investigation activity.

Elaboration: Once a system solution has been identified at the inception level, the elaboration phase assesses the requirements of the system in a more detailed manner. The technical requirements such as the architecture and the components of the envisioned system, and the expected expenditure on the project are established. A project plan which should include any significant potential risks, a requirements specification model in the form of use cases and a specific software development plan should be the deliverables of this phase.

Construction: This phase is concerned with designing the system, writing the codes for the software and testing the system to ascertain that it meets the requirements. The deliverable of the construction phase is a functional application accompanied with the requisite documentation.

Transition: This final phase of the RUP process deals with what can be described as going live. This basically means that the developers hand over the system to the end user. In the case of public domain software, this is the stage at which a product is released to the market. A number of tasks are involved in making sure that the system runs as was expected in the real user environment and that the users are able to use it.

RUP process is made up of six core workflows at the top and three supporting workflows at the bottom respectively. The core workflows are also referred to as engineering workflows because they are concerned with the actual design and building or construction of the system. Some authors omit business modelling and deployment workflows when discussing RUP engineering workflows (Avison and Fitzgerald, 2008). As we shall see in a little while, this is not advisable given the role the two workflows play in RUP process. A brief description of each of the workflows is given here below:

Business modelling: The main goal of business modelling workflow is to demonstrate how the system will function from a business perspective. This is achieved by developing business use case models. This helps to give a better understanding of the business problem for which the system is to be developed, and enhances the chances of the success of the project as the design is likely to be more accurate.

Requirements: The requirements workflow is aimed at developing system requirements specification document. It entails establishing and defining the scope or boundaries of the system, specific system functions as well as identifying system actors. Actors are entities that will interact with system in one way or the other. A use case capturing all these aspects is drawn to create a system requirement.

Systems analysis and design: The analysis and design workflow is concerned with the logical design of the system. It translates the system requirements model developed under the requirements workflow into a specification that programmers can use to write codes for the system. This workflow ignores the non-functional requirements and focuses on the functional requirements of the system. The laborious and tedious, but important tasks of defining and fine tuning the system architecture, database design and analysing the functional features of the system are performed at this workflow.

Implementation: Writing codes for the components of the system as per the specification developed at the analysis and design workflow is done at the implementation workflow. A

careful planning of how the various parts are to be developed is a critical success factor at this level. The developed components are tested individually as well as partially integrated systems which are in turn developed in an evolutionary manner until a complete system is developed.

Test: The goal of the testing workflow is to make sure that the system developed has no flaws or defects. The completed system is tested as integrated whole to find out if it meets the requirements and performs its intended function. While most testing is done at the test and implementation workflows, it is worth noting that testing is an integral part of the system development process under RUP model that is performed iteratively at all levels. The nature and extent of the tests varies at the different stages.

Deployment: The deployment workflow deals with transferring the system from the developers to the owners. The exercise will largely be determined by the type of the system to be deployed. Nonetheless, typical activities at this stage include but are not limited to installing and testing the software in its operational environment, conducting end user training and change over from the manual or existing application to the new one (Avison and Fitzgerald, 2008).

Configuration and change management: Development of information system is normally a long and an enduring process involving many changes and delivery of numerous artefacts. In order to have a smooth running process and deliver a system that meets quality metrics, there must be a clear mechanism to manage the many changes. This is what this supporting workflow does.

Project management: Project management is also a supporting workflow that manages the entire software development process. RUP provides for a phase centric plan as well as an overall project plan to ensure that risks are properly managed, quality targets are achieved, timelines are met, cost overruns are avoided and requirement creeps are kept under the wraps if they occur.

Environment: This final workflow's objective is to guarantee that the development team have access to the necessary software tools and the supporting accessories such as relevant processes and methods it requires to successfully develop the software.

A discussion of the RUP methodology is never complete without mentioning that it is anchored on six sound software engineering best practices aimed at competent management

of software projects in order to deliver software that meets acceptable quality standards. The six best practices are: develop software iteratively, manage requirements, use component-based architectures, visually model software, verify software quality and control changes to software. These best practices have been partially captured under the phases and workflows sections of the RUP process.

RUP is certainly not appropriate for all software projects but it has features that can be beneficial to many information system projects. Its emphasis on documentation and change control and management can be of great value to large software projects where many changes are always inevitable. As reflected by the use of workflows and the phased approach, the RUP methodology also looks at software development as an undertaking that involves business processes, people and organisational as well as technical issues. This reduces chances of software project failures. The use of iteration and incremental development makes the RUP methodology very flexible and helps to ensure that quality software is delivered as faults are discovered and corrected earlier in the development process. Last but not least, RUP methodology makes use of code reusability. The net effect of this is increased productivity of developers and quicker delivery of software products.

On the flip side, the RUP process is a fairly complex one and can be difficult to understand by developers. Producing desired results with RUP methodology requires highly skilled and experienced software development personnel to be in charge of an information systems project.

3.3 Rapid Application Development

Rapid Application Development (RAD) is a modern system development methodology that has gained popularity among software developers. The term RAD was coined by James Martin in 1991 (Britton and Doake, 2006). RAD was a response to the inherent weaknesses of traditional development methodologies which focused more on processes, procedures and technical design at the expense of user needs, time and in most cases cost. It takes iterative and incremental or evolutionary development approach to systems development. This is in line with one of its tenets of software development that views system's requirements as a dynamic and an evolving phenomenon rather than a static one. The implication of this is that the users of the system are involved throughout the development life cycle, leading to delivery of software products that meet user requirements. RAD often uses evolutionary prototyping technique to achieve this.

One of the most important aims of RAD is to develop high quality software quickly using as minimal resources as is practically possible. RAD achieves this by splitting the application to be developed into several small units, each of which is allocated a specific time frame within which it must be developed. Since RAD advocates for strict adherence to development time frames, deadlines for delivery of the various components (also known as timeboxes) cannot be extended. RAD prioritises system requirements in order of their importance to the system from the most to the least critical. The most important requirements are timeboxed, developed and delivered first to the user. The process continues until all the necessary features of the system are developed. The prioritisation exercise is normally conducted in a formal workshop comprising all the system stakeholders including users.

RAD and prototyping have inspired the development of a new generation of software development methodologies which are better organized and developed (Sommerville, 2004). These new methodologies are collectively referred to as Agile Methods, and include Dynamic Systems Development Method (DSDM) and Extreme Programming (XP) among others. DSDM retains most of the characteristics of the original RAD methodology. It is also the most popular of this new crop of methodologies.

Although RAD is not a very process or procedure intensive methodology, its processes are organised around four phases of requirements planning, user design, construction and cutover. The short life cycle of RAD contrasts with that of many traditional development methodologies which normally have at least six stages. The four phases of RAD are discussed in the following paragraphs.

Requirements planning: This is the first stage in a RAD development and is intended to define the system requirements. All the system users, developers and senior management are involved at this stage. The users give information that is translated into system requirements by the analyst. Participation of senior management is important because information systems development is not just a technology problem; it is a business issue as well. Developers need to get commitment from senior management of their support for the project. Avison and Fitzgerald (2008) say that Joined Requirements Planning (JRP) and Joint Application Development (JAD) techniques are used in the requirement planning. The former is a workshop involving senior management and developers for purposes of formulating and agreeing on requirements of the proposed system from the perspective of the executive. The latter is a facilitated workshop that consists of all users of the system including a

representative of senior management to gather and define system requirements. There can be a number of such workshops depending on the size and nature of the proposed system.

User design: In RAD development, user design is meant to translate the system requirements defined during the planning phase into specifications that can be implemented by programmers. A series of JAD workshops involving experienced users are organised to accomplish this mission. Developers use CASE tools to build prototypes of parts of the system, particularly those that are contentious to demonstrate how the system will look like. These prototypes can then be developed incrementally and fine tuned to the satisfaction of users and developers (Britton and Doake, 2006).

Construction: RAD's construction phase converts the user designs into actual code that can be executed by the computer. Any prototypes developed during the user design phase are refined by the developers. The construction phase involves selected users and experienced programmers and information systems professionals using specialised software tools. Avison and Fitzgerald (2008) refer to these experts as skilled with advanced tools (SWAT) teams. Developers test the completed components as well as the entire system to ensure that both the functional and non-functional requirements are met. The requisite documentation is also done at the construction level in a RAD development.

Cutover: This is the final stage in a RAD project whose main objective is to get the system working in the owner's IT infrastructure. It includes a final round of system testing using live data, user training and changeover. Any organisational changes occasioned by the introduction of the system are also resolved and addressed at this level. The changeover approach used often varies from one organisation to another as dictated by the nature of the systems (old and new), size of the new system and operational realities of the organisation and so on and so forth.

Using RAD in information systems development offers many benefits to system developers. Timeboxing enables developers to deliver systems faster. This often reduces the cost of developing systems considerably. Its emphasis on user involvement throughout the development life cycle has twofold effects. One, the system delivered is most likely to meet the user needs closely. And two, chances of users resisting the new system are very low since they will have interacted and evolved with it over time. Another key benefit of RAD is that codes and modules are reusable. This makes it easy to add new components to the system if changes in the business requirements occur.

All its strengths notwithstanding, RAD is not a panacea to all software development problems. Its focus on speedy software development coupled with the rigidity imposed on the process by timeboxing may compromise quality of the delivered software. The project team may leave out some important features in order to adhere to strict delivery timelines. RAD projects can only succeed if managed by highly skilled system development professionals. In some cases this may push development costs high. RAD development methodology is appropriate for small and medium sized information systems where documentation is not a crucial requirement.

3.4 Justification of the chosen methodology

As was mentioned in the introduction of this chapter, this project will use RAD as the main methodology but aspects of waterfall and RUP methodologies will also be used whenever is appropriate. RAD's iterative and evolutionary approach is the main reason why it has been chosen for this project. The evolutionary prototyping technique will particularly be useful in building the application. Besides, RAD's timeboxing technique will certainly help to minimise the development time and cost while still ensuring that software that meets the specification requirements is delivered. Apart from its obsession with technical details geared towards developing a robust software product, the other main reason for using waterfall in this project is its emphasis on detailed documentation of the development process. A good documentation makes it easy to maintain software. While RUP also uses iterative and incremental software development approach, its use in this project will be due to its focus on business requirements of the system. RUP uses business use cases to deliver systems that closely meet the user requirements.

3.5 Conclusion

The unchanging goal of any software development project is to deliver software that meets its functional and non-functional requirements within the budget and in time. The use of methodologies makes it feasible to achieve this goal by bringing order to the software development process. That said, it is important to note that there is no universal methodology in software development. The choice of methodologies used in a software project is influenced by many factors.

Traditional methodologies are suitable for large software projects which have fairly stable requirements. Such projects also require enormous amount of documentation and prior

planning, requirements that are easily met by the traditional methodologies. Modern software approaches on the other hand are generally suited for projects which have constantly changing requirements and which need greater user involvement, communication among project team members and early delivery of a working software code. Nevertheless, many software projects still involve the use of both traditional and modern methodologies. It is worth noting though, that RAD and agile development methodologies are becoming increasingly popular with software developers.

Chapter Four

Requirements Analysis

4.0 Introduction

Requirements analysis is an important phase in a software development process. The quality of software delivered often depends on how well the activities of this phase are carried out. It involves identifying and eliciting, describing and specifying and validating system requirements. Another important aspect of this phase of the software development project is the identification of people who will use the system and what their roles will be. The specific roles for the proposed online reservation are outlined under the roles section below.

4.1 Roles

Many people will use the system for one purpose or the other. However, the roles that have been discussed below are only for personnel whose usage of the system will be critical if it has to meet the needs for which it is being developed. Without following the order of their importance, the roles include:

Applications Administrator: The role of the applications administrator will be to manage user (customers or staff of the hotel) accounts, ensure the security of the system by restricting users to appropriate access privileges and escalating any security threats to the system beyond his or her mandate to the systems administrator or network security personnel, update the website in case the hotel has new content that needs to be added to the web service as well as remove outdated content. Lastly, the applications administrator will also provide user support to any person using the system.

Receptionist: Any person holding this position will access the room booking facility to be able to answer queries from customers who make enquiries on phone or come personally to the hotel. This role will not have privileges to make changes to the records but will be able to use queries to get the information he or she needs from the system.

Reservation Manager: The Reservation Manager will have periodic access to the sales facility of the system to enable him or her generate reports on sales of the various services offered by the hotel. He or she will not be able to update any records. Nevertheless, he or she

will have the privilege to use a wide range of queries to extract information he or she may need from the system.

Customer: The customer is the most important actor in this system. The goals of the other three actors only support the customer in achieving his or her goals in the system. The customer will create an account, reserve a room and give feedback on their experience of using the system. He or she will also be able to buy tickets of transport service providers from the hotel's website.

4.2 Initial Functional Requirements

The system to be developed will deliver the following specific functionalities:

1. The system will allow users to create accounts online
2. The system will have an email notification facility that will automatically be sent to users upon completing a task such as creating or deleting an account successfully.
3. It will have a chat facility that customers can use to make enquiries in an interactive manner with hotel staff
4. The system must have a link facility on the hotel's website that when clicked takes the customer to a list of services offered by other service providers such as cab companies, bus companies, airlines and event organisers among other providers.
5. The system must be able to generate various types of reports for the hotel.

4.3 Data Gathering

Data gathering refers to a set of activities that the developer or analyst carries out to put together the requirements of a system. Some authors refer to it as requirement elicitation. The information obtained during the exercise helps the developer to get a clear picture of the customer's expectation of the system. The principal aim of data gathering therefore is to ensure that the requirements of the system to be developed are captured clearly, accurately and completely so that the end product of the process meets the purpose for building the software as closely as possible. Data gathering requires the developer to interact directly with the customer or the would be end user of the system.

There are several techniques that can be used to gather information system requirements. However, in this project, the questionnaire technique has been used mainly because it provides an opportunity to reach a larger number of respondents from different hospitality

firms. The other reasons for using questionnaire include the ease, speed and low cost with which it can be administered.

4.4 Analysis of questionnaire

The questionnaire was administered in medium size and big hotels within Nairobi. The aims of the questionnaire were to: establish whether hotels were sharing information in a manner that could lend itself to a web service solution, find out if any hotels are interested in or are already using web services to market and sell their services and find out what key non-functional features hotels would like a web service to have if they were to implement it. A total of ten respondents from six hotels in Nairobi completed the questionnaire. The results of the investigation are shown in the bar graphs below:

Table 1: Response of hotel employees in Nairobi on whether they think their companies should implement web services

Web Service Survey				
	May Need Web Service	Don't need Web Service	Not Sure	Total
Hotel staff	7	1	2	10

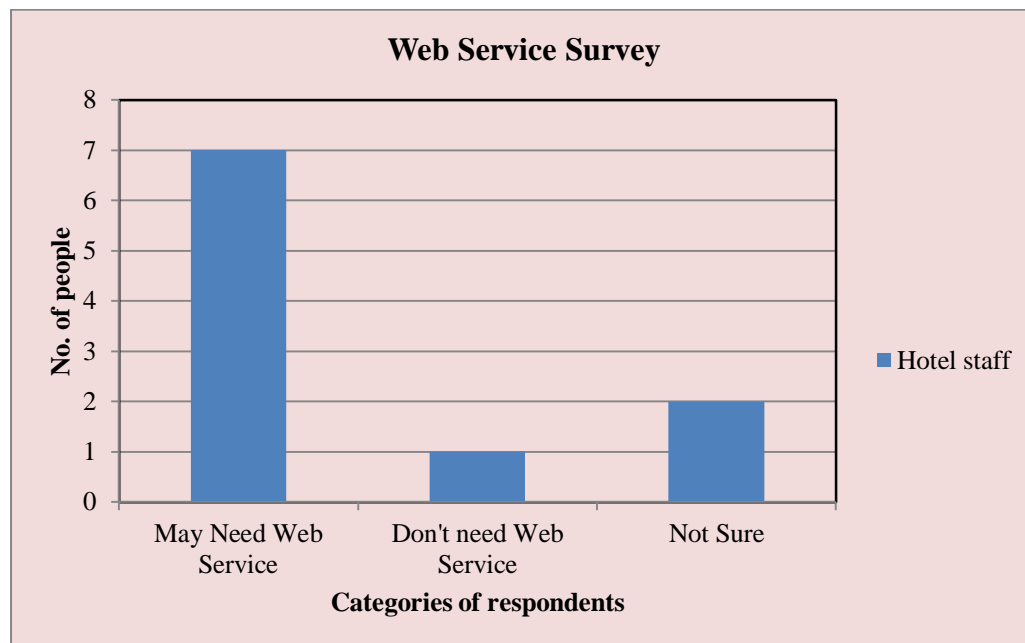


Chart 1: Comparison of desire by staff of mid size and big hotels in Nairobi to implement web service

Table 2: Non-functional requirements that hotel employees in Nairobi would like to see being implemented in a web service

Non-functional requirements users want in a web service				
Requirement	Timeliness	Security	Confidentiality	Accuracy
Hotel staff	6	10	7	10

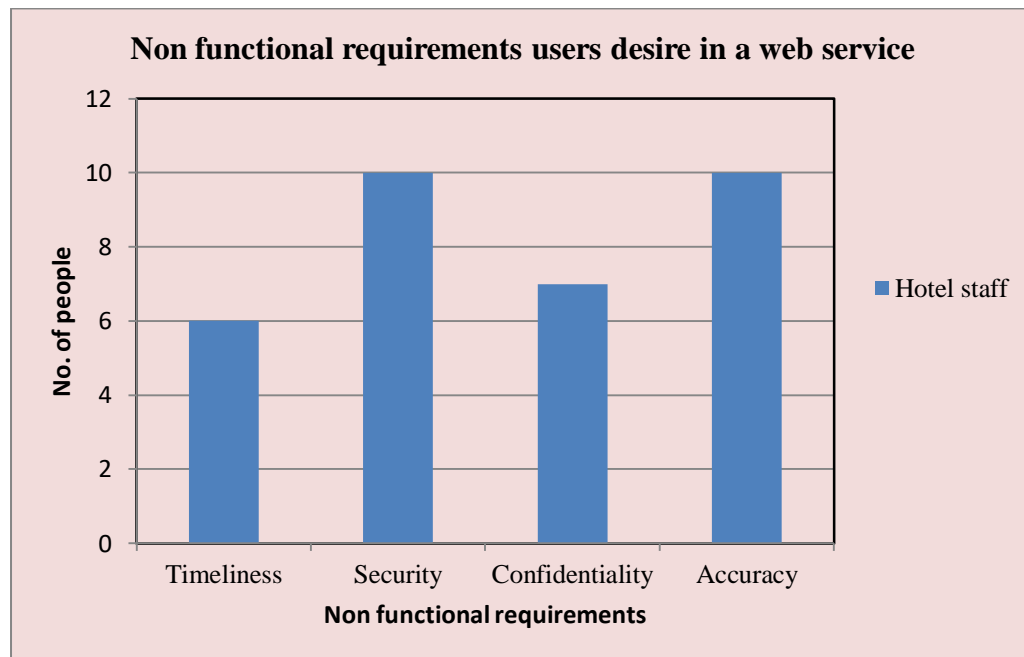


Chart 2: Functional requirements identified by hotel staff in various Nairobi hotels as necessary in a web service.

4.5 Detailed functional requirements

Functional requirements specify what a system shall or will do once its development is completed. A good system must be able to perform the tasks and operations outlined in a functional requirements specification document. The functional requirements specification document acts as a reference document for system designers. It is thus not uncommon, particularly in modern system development environments to find developers moving back and forth between this stage and the design stage of system development. The following sections discuss the functional requirements of the reservation web service system.

Functions: The system shall perform the following specific functions:

- The system shall allow the applications administrator to add new users and/or delete users existing users.
- The applications administrator must be able to grant users of the system appropriate access privileges.
- The system must allow the reservation manager to view the number of bookings made by the hotel.
- The receptionist must be able to confirm the room availability using the system as this will be useful to him or her when responding to customers' enquiries on telephone or via email.
- The system shall allow customers to sign up before accessing the hotel services
- The system must allow the customer to see the rates charged by the hotel.
- The system shall have a facility to enable applications administrator to cancel booking that has been made. Customers must therefore send a cancelation request by email to the administrator in order to cancel a booking. An email notification will be automatically sent to the customer immediately the booking is cancelled.
- The customers shall have an opportunity to give feedback on the services of the hotel at the end of their stay.
- The system shall allow the customer to access and call a taxi service, buy bus or airline tickets while on the hotel's website.
- The reservation manager and the applications administrator will be able to view customer feedback, but only the applications administrator can delete customer feedback.

Reports: The reservation manager shall be able to generate various reports that he or she will share with the general manager, and sales and finance departments as and when required. The reports should be able to give the following information:

- Monthly, quarterly and yearly number of bookings classified according to the nationalities of customers.
- Monthly, quarterly and yearly number of bookings classified according to branches of the hotel.
- Monthly, quarterly and yearly reports of the customers who accessed services of other hospitality service providers from the hotel's system.

4.6 Use case diagrams

Use case diagram is a requirements analysis technique that analysts and/or system developers often use to present the system from the perspective of the user. It has three important components namely, actors, use case and system boundary. Actors are entities that are affected by or interact with the system in one way or the other, and can be people, organisations, systems or subsystems. Use cases represent the actual functional requirements of the system and are always connected to the actors using a line which represents a relationship between them. These functional requirements (use cases) together make up a system. The system boundary separates the use cases from the actors. Figure 2.6 shows the use case diagram of this system.

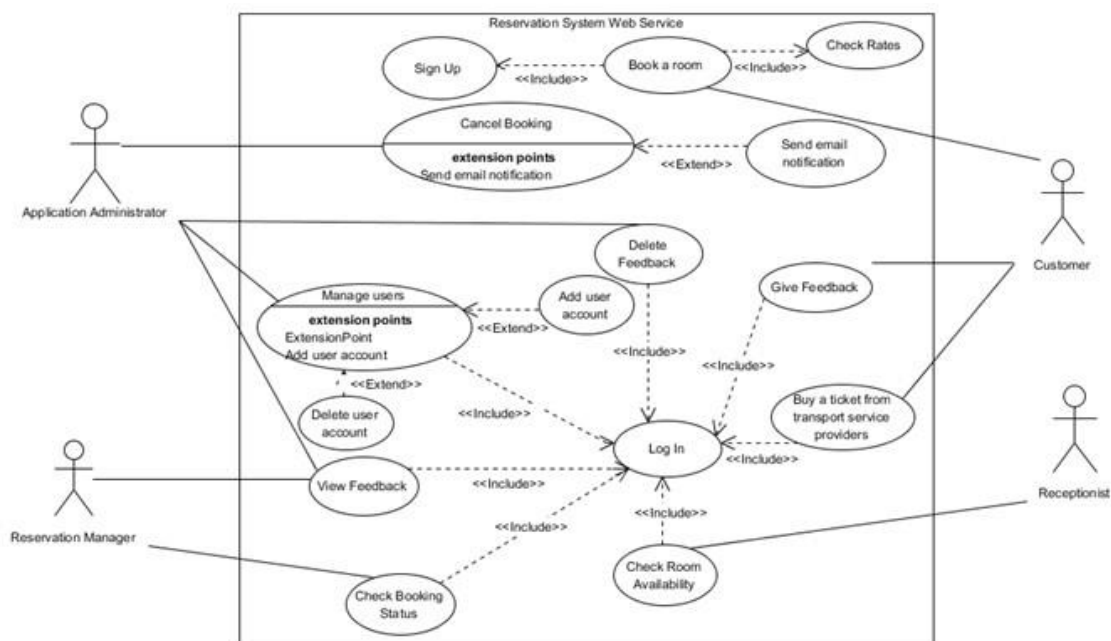


Figure 4.1: Use case diagram of the reservation system web service.

4.6.1 Use Case Diagram Documentation

Use case documentation involves stating the sequence of actions performed by a system in order to achieve its goals. Documenting use cases aids in understanding what the system intends to do. Developers can thus use it to communicate among themselves and with end users regarding the functionalities of a system being developed. The use case diagram in figure 2.6 is documented in the paragraphs that follow.

Use Case One: Book a room.

Primary Actor: Customer

Primary Scenario

Precondition: User has successfully signed up.

Flow of Events

1. The use case begins when customer opens a browser and opens the hotel's website.
2. Customer creates a user account.
3. Customer receives email notification from the system.
4. Customer activates the account.
5. Customer checks room types and rates.
6. Customer completes a room reservation form.
7. Customer logs out of the system and use case ends.

Secondary scenario

- 1a. Access to the hotel's website fails, use case ends

Post condition: A room is booked by the customer.

Use case Two: Give Feedback

Primary Actor: Customer

Primary Scenario

Precondition: The customer booked a room and has used the hotel services.

Flow of Events

1. The use case begins when customer opens a browser and opens the hotel's website
2. Customer logs into the system
3. Customer opens submit feedback web page
4. Customer writes and submits feedback
5. Customer logs out, and use case ends

Secondary Scenario

- 1a. Access to the hotel's website fails, use case ends

- 4a. Customer fails to submit feedback due to lack of internet connection
 - 4a1 Customer logs out and use case ends.

Post condition: Feedback is given or customer declines to give feedback

Use Case Three: Buy a ticket from transport service providers

Primary Actor: Customer

Precondition: The customer has successfully logged in to the hotel's system

Flow of Events

1. Use case begins when customer opens browser and accesses hotel's website.
2. Customer logs in.
3. Customer click link to transport service providers website.
4. Customer browses transport service catalogue.
5. Customer selects transport service.
6. Customer reserves a ticket and use case ends.

Secondary Scenario

- 1a. Internet connections fails, and use case ends.

- 3a. Transport service providers website fails to open.
 - 3a1. Customer tries again or user gives up and use case ends.

Post condition: A ticket is bought or a taxi is called by the customer.

Use Case Four: Check Room Availability

Primary Actor: Receptionist

Primary Scenario

Precondition: Receptionist has received enquiry from customer.

Flow of Events

1. Use case begins when receptionist receives an enquiry from customer through email or telephone.
2. Receptionist opens browser and accesses hotel's website.
3. Receptionist logs in
4. Receptionist opens room reservation web page
5. Receptionist browses rooms catalogue, confirms availability of room and use case ends.

Secondary Scenario

2a. Browser fails to open, receptionist tries to open it again or use case ends.

Post condition: Room availability is confirmed

Use Case Five: Check Booking Status

Primary Actor: Reservation Manager

Primary Scenario

Precondition: Customer bookings exist.

1. Use begins when reservation manager opens browser and hotel's website
2. Reservation manager logs in
3. Reservation manager opens room reservation page
4. Reservation manager opens reservation report section of the room reservation web page, use case ends.

Secondary Scenario

1a. Browser fails to open, reservation manager tries again or use case ends.

Post condition: Customer booking report displayed.

Use Case Six: View Customer Feedback

Primary Actor: Reservation Manager

Precondition: Customer feedback exists

Flow of Events

1. Use begins when reservation manager opens browser and hotel's website
2. Reservation manager logs in
3. Reservation manager opens room reservation web page
4. Reservation manager opens customer feedback section of the room reservation web page, use case ends.
5. Reservation manager opens reservation report section of the room reservation web page, use case ends.

Secondary Scenario

1a. Browser fails to open, reservation manager tries again or use case ends.

Post condition: Customer feedback is displayed.

Use Case Seven: Manage Users

Primary Actor: Applications Administrator

Precondition: Administrator has logged into the system

Flow of Events

1. Use case begins when applications administrator opens browser and hotel's website.
2. Applications administrator logs in.

3. Applications administrator opens user accounts web page.
4. Applications administrator views list of users and selects a user account name.
5. Applications administrator performs appropriate task.
6. Applications administrator confirms action and use case ends.

Secondary Scenario

- 1a. Browser fails to open, applications administrator tries again or use case ends.
- 5a. Applications administrator selects a wrong user account name.
 - 5a1. Applications administrator does not click cancel, selects the right user account name in step 4 and proceeds to steps 5 and 6, use case ends.

Post condition: Administrator performs appropriate action.

Use Case Eight: Cancel Booking

Primary Actor: Applications Administrator

Precondition: Customer booking exists

Flow of Events

1. Use case begins when applications administrator opens browser and hotel's website
2. Applications administrator logs in
3. Applications administrator opens room reservation web page
4. Applications administrator opens list of customers who have made booking
5. Applications administrator selects booking of a customer to delete.
6. Applications administrator click cancel button, confirms cancelation and use case ends.

Secondary Scenario

- 1a. Browser fails to open, applications administrator tries again or use case ends.
- 5a. Applications administrator selects a wrong customer booking for cancellation

5a1. Application administrator does not click cancel, selects the right customer booking and proceeds to number 6, use case ends.

Post condition: A customer booking is cancelled.

4.7 Non-functional Requirements

Non-functional requirements are nearly as important as the functional requirements. They are desirable qualities that determine whether the software developed is of acceptable quality. It is important to note that the successful implementation of the functional requirements largely contributes to the non-functional requirements. Non-functional requirements include but not limited to: reliability, usability, security and availability. Each of these requirements is discussed in the subsequent paragraphs.

Reliability: Reliability is the attribute of software that enables it to behave in a consistent manner as it performs functions as stated in its requirements and design specifications. A reliable system gives users the confidence that they will not experience unexpected behaviour from the system during its operation. Users want to be assured for instance that the system will not automatically restart or abort before completing processing their work. In this system, users will always get automatic email notification upon cancellation of their booking or deletion of their account by the applications administrator. Again, as long as there is internet connection, customers will always be able to connect to transport service providers website if they so need.

Usability: The usability non-functional requirement is one of the most important requirements that any good software product should meet. Usability defines the ease with which the user is able to perform tasks for which the software was intended. For this to be achieved, users must be able to learn how to use the system. The amount of effort the user needs to put into learning the software to be able to use it efficiently is thus an important indicator of how usable the software is. While the technical features of this system will be given proportionate attention, a lot more effort and attention will be devoted to making the system as user centric as possible. The system will have graphic user interface (GUI). Users will access the system through a web browser. The user interfaces will be designed to enable users to navigate easily around the system.

Security: This requirement aims to ensure that users who are not authorised to use the system are not able to access it. Better still, authorised users will also be assigned appropriate

permissions so that the integrity of any confidential data is maintained. This system will require users to have accounts in order to access and use it. User activities in the system will thus be determined by their class or category to ensure that the system security is not compromised.

Availability: This non-functional requirement deals with issues such as the frequency of system down times and how long it takes the system to resume normalcy in case of a down time. To be more specific, system availability describes the amount of time the system must be up and running for use by end users. A good system is one that is able to serve users especially during peak times. This system will basically run on the internet. However, its availability will not just be measured by the availability of the internet, but by the ability of transport service providers to publish their services and whether customers will always be able to access those services. With an uninterrupted internet connection, the system will provide 99.9% availability to users.

4.8 Conclusion

Requirements analysis stage of the system development process is a very demanding exercise. And though it is fairly technical and requires that the analyst or the developer possesses good technical skills, it should be noted that it also requires a high degree of social engineering skills (Group and Science, 2006). The reason for this is that it often requires the highest level of stakeholder involvement in the development process. Regardless of the techniques used in the requirements specification development, the key objective of the activity is to turn the desires of the stakeholders into requirements that can be implemented. But even more important is the need to develop a requirement specification document that closely meets the needs of the user. In this project, the functional requirements have had to be changed a number of times before coming up with the final set of requirements. It also emerged that the functional and non-functional requirements are intertwined. Non-functional requirements put demands on the system that must be met by the functional requirements.

Chapter Five

System Design

5.0 Introduction

The system design stage concerns itself with converting the requirements specification into a model that can be implemented. It involves producing specifications for databases, data structures, processes and user interfaces as seen by the end users (Gould, 2008). Though the traditional approach to system design differs significantly from the modern approaches, the ultimate goal of system design is still to construct a model that can be used to develop a system that meets the user requirements as closely as possible. This fact has been taken into account in the design of this system.

5.1 Database Design

Database design occupies an important place in the system design process. This is because data that users will need to solve their various problems is stored in the data. The other parts of the system only facilitate the process of using this data to meet user needs. There are several design techniques used in database design. However, in this project, entity relationship diagram (ERD) has been used as the main design technique. Data modelling using ERDs begins with conceptual design, then progresses to logical design and ends with physical design. The subsequent paragraphs give further details on each of the three ERD approaches.

5.1.0 Conceptual Design

A conceptual database design is concerned with presenting the system from the perspective of the user or business. It shows how the various high level functionalities are related to each other without delving into the technical aspects of the design. That is, a conceptual design shows entities and their relationships without the accompanying attributes. A conceptual data model or Entity Relationship Design (ERD) is the simplest design document that technical people can use to demonstrate the system to the end user. Figure 2.6 shows the conceptual ERD of the database for the online reservation system.

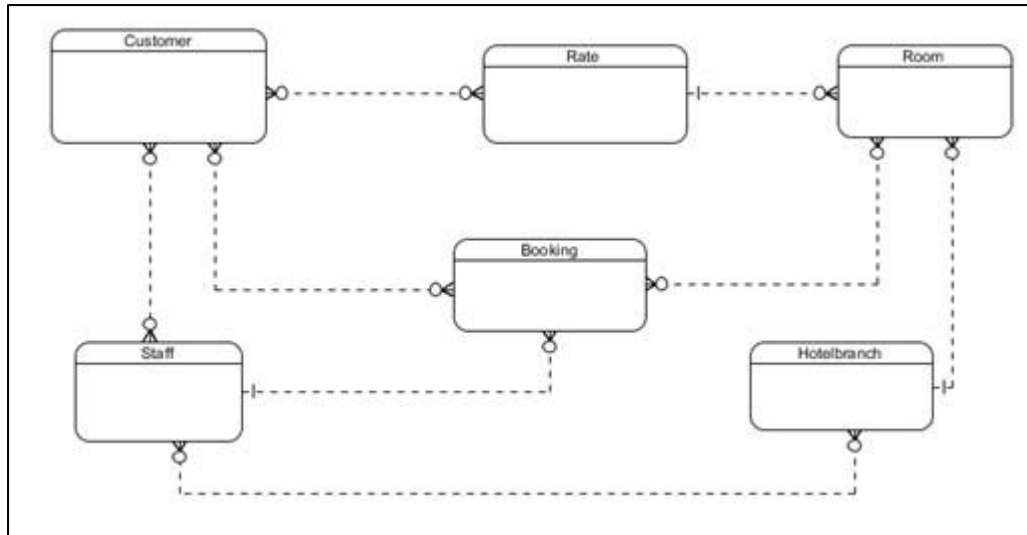


Figure 5.1: Conceptual ERD for the online reservation system database

5.1.0 Logical Design

A logical design or data model presents the system in a more detailed manner than the conceptual design. At this level, the ERD includes all entities in the database and attributes in each of them. The key attributes, that is, the primary key and foreign keys for each entity and the relationships between entities are also specified in the logical ERD. It is worth noting that a logical ERD is a technical document that is only used to communicate the system functionalities among the technical members of a system development team. Simply put, a logical design translates the business requirements depicted by the conceptual design into a technical design that when improved, can be used to implement a physical system. The logical ERD of the reservation system database is shown in figure 2.7.

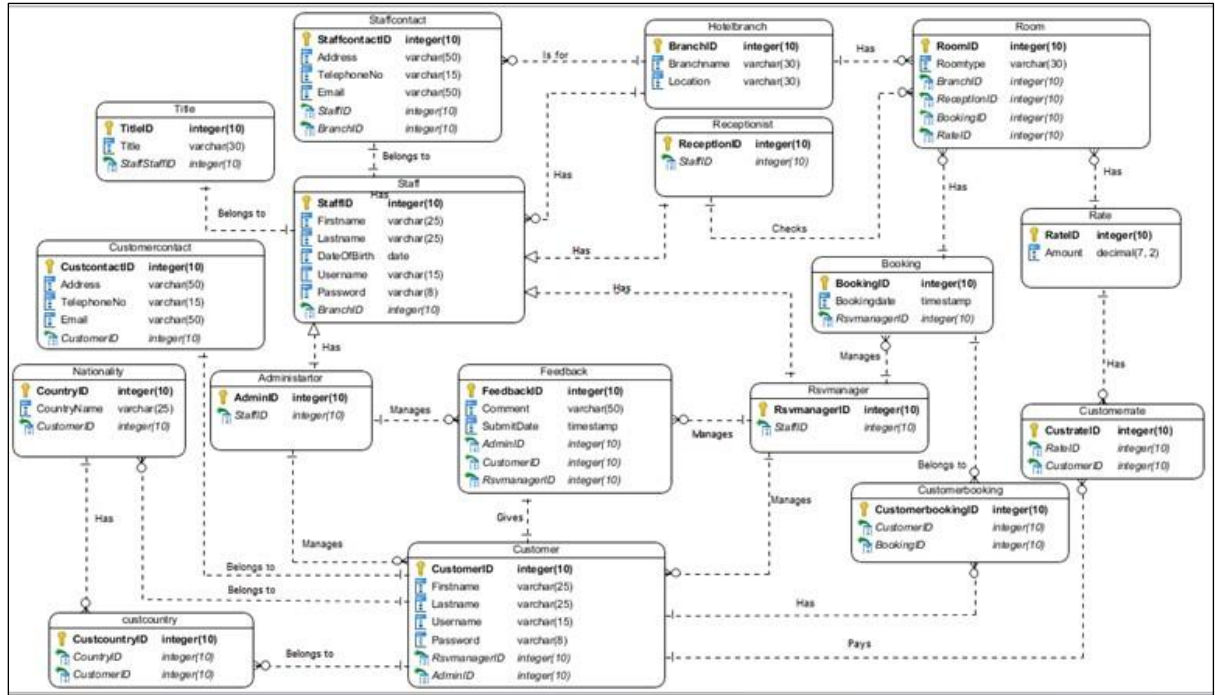


Figure 2.9: Logical ERD for the online reservation system database

5.1.1 Physical Design

The physical design model goes further than the logical design model to show how the system will be constructed in a database. At this design level, the data types and length of each attribute of all the entities are specified. The key attributes (primary and foreign keys) are also clearly specified. Like the logical ERD, the physical ERD clearly shows the relationships between and among entities of the database. For all practical purposes, the physical ERD is a technical document that can only be understood by technical people in the software development team. The database developers use it to build the actual database. The user interface designers, frontend and backend developers refer to it as well. Figure 2.8 illustrates the physical ERD of the online reservation system.

5.2.1.1 Screens

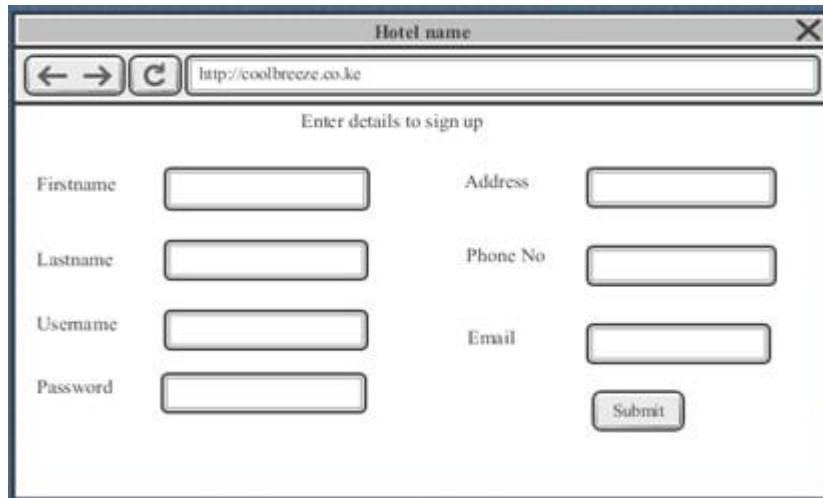
The following sections show a number of screen layouts through which users will interact with the system. These high level interfaces (some low level interfaces are not shown) include the homepage, sign up page, sign in page, room reservation page, feedback page and the web service page. Each of these interfaces is briefly explained in the succeeding sections.

Homepage: The homepage is the first page that appears when the user enters the universal resource locator (url) of the hotel's website. From here, a user can visit other parts of the system that he or she is permitted to access.



Figure 5.2: Website homepage

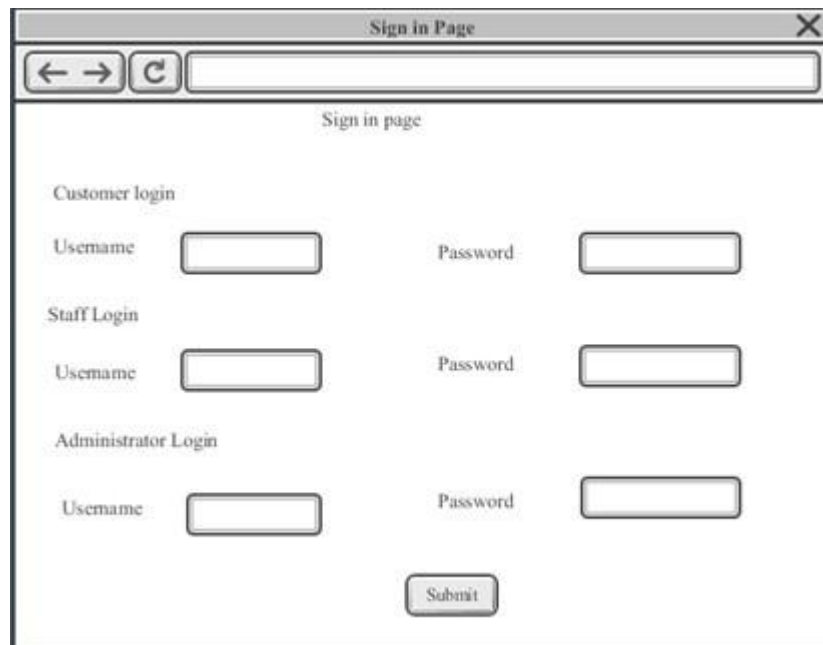
Sign up: The sign up interface is used by new users to create accounts that they will use to access the system.



The screenshot shows a web browser window titled "Hotel name". The address bar contains "http://coolbreeze.co.ke". Below the address bar, the text "Enter details to sign up" is centered. The form consists of two columns of input fields. The left column contains fields for "Firstname", "Lastname", "Username", and "Password". The right column contains fields for "Address", "Phone No", and "Email". A "Submit" button is located at the bottom right of the form area.

Figure 5.3: Sign up page

Sign in: This interface will be used by users to login into the system. The interface has three distinct sign in slots since each of the three categories of users have different access privileges to the system.



The screenshot shows a web browser window titled "Sign in Page". The address bar is empty. Below the address bar, the text "Sign in page" is centered. The form is organized into three sections: "Customer login", "Staff Login", and "Administrator Login". Each section contains a "Username" input field and a "Password" input field. A "Submit" button is located at the bottom center of the form area.

Figure 5.4: Login page

Room reservation: The room reservation interface will be used specifically by customers to reserve a room of their choice. The combo boxes for room type and room rate have a drop

down list from which the customer can select a room type and its corresponding cost respectively.

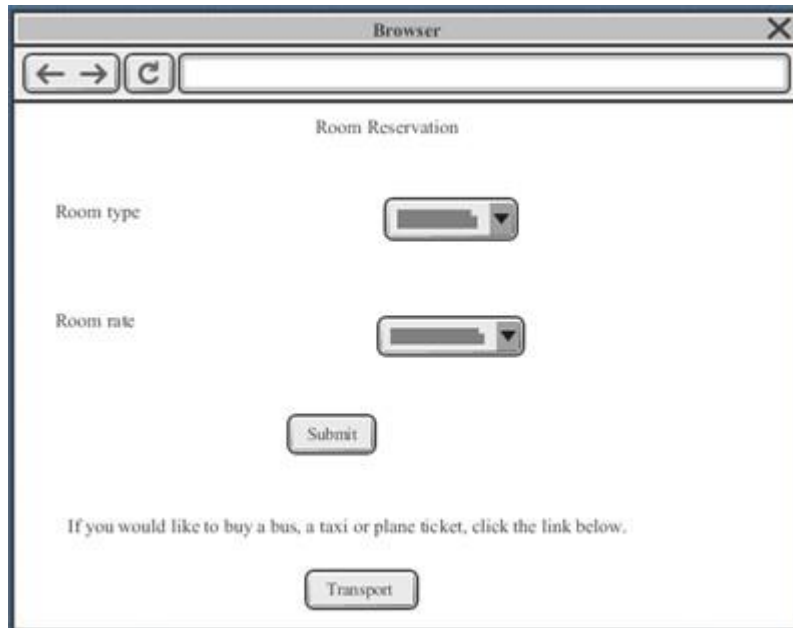


Figure 5.5: Room reservation page

Feedback page: The feedback interface will be used by customers to give their comments regarding the services offered by the hotel. The applications administrator and the reservation manager will also use this interface to view a customer's feedback. The applications administrator can go a step further and delete a customer's feedback from this interface.

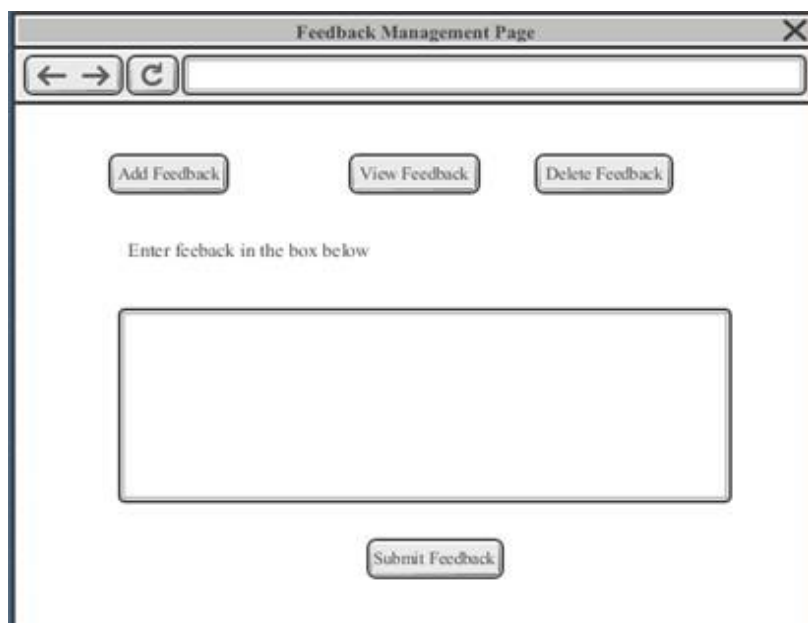


Figure 5.6: Feedback page

5.3 Data Flow Diagrams

A Data Flow Diagram (DFD) is a structured modelling technique used to show how data flows between the different processes within a system. As opposed to use case diagrams that present the system to users from a functional point of view, DFDs present the system to users as a set of interrelated processes. One main advantage of using DFDs in modelling a system is that it is simple and therefore easy to understand by non-technical people. This makes it a good tool for explaining to clients what a system will do. The DFD of the proposed system is shown in figure 5.7 below.

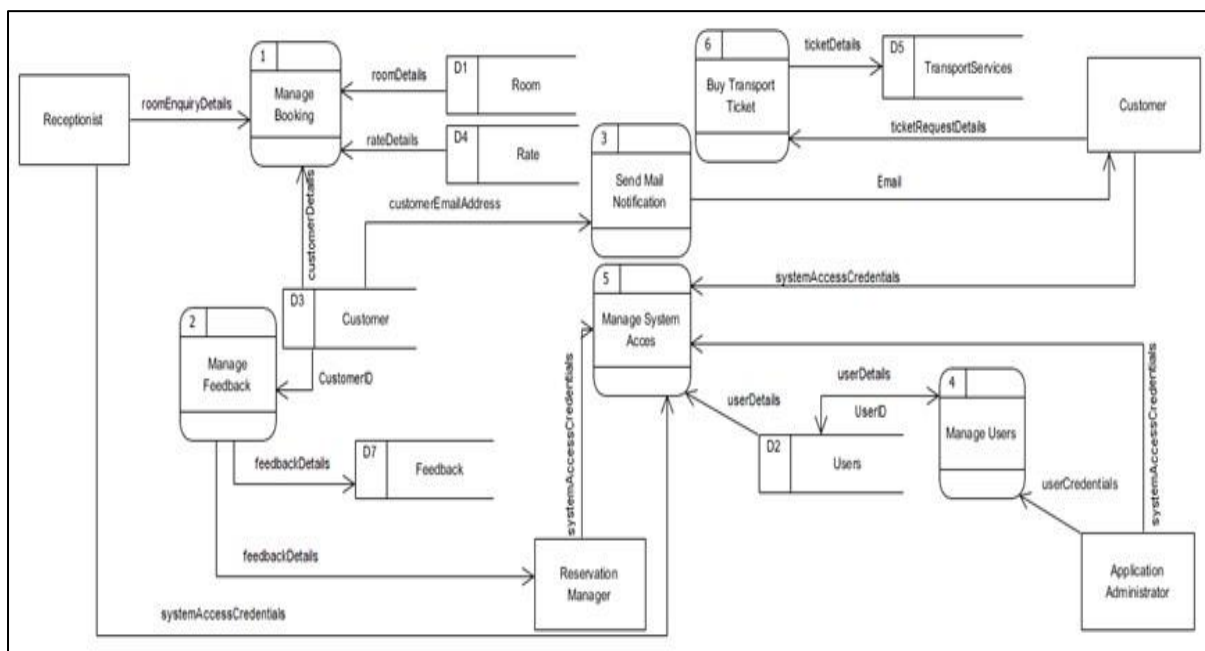


Figure 5.7: DFD for the Reservation System Web Service

5.5 Conclusion

This chapter marks the end of the design activities of this project. An important aspect of the system design stage of software development that was brought to the fore during this process is that it is closely tied to the requirements analysis, particularly the system’s functional requirements. A few changes had to be made on the functional requirements as well as the use case diagrams after developing the logical and physical ERDs. This reaffirmed the often stated fact that system development in modern times is ideally an iterative process that requires flexibility and agility.

The artefacts of the design stage forms the basis of the implementation stage of this software development project. The physical ERD will be implemented as the underlying database.

The user interface design models represent the web pages that are a crucial part of this system. Use case diagrams and DFDs provide a constant reminder to the developer as to how users view the system. Codes will be written to develop the various web pages that make up the websites. Besides, there will be server side codes that will connect the website and its interfaces to the reservation database as well as link the database to the web service that allows customers to access the services of transport service providers from the hotel's website.

References

- Avison, D. and Fitzgerald, G. (2008) *Information Systems Development: Methodologies, Technologies and Tools*. 4th edn. London: McGraw-Hill.
- Britton, C. and Doake, J. (2006) *Software System Development: A gentle Introduction*. 4th edn. London: McGraw-Hill.
- Despa, M. L. (2014) 'Comparative Study on Software Development Methodologies', *Database Systems Journal*, 5(3), pp. 37–56. doi: 10.1109/MAHC.1983.10102.
- Du, G. (2004) *Web Services Performance Study*. University of Leeds. Available at: https://minerva.leeds.ac.uk/bbcswebdav/orgs/SCH_Computing/MSProj/reports/0304/du.pdf.
- Dustdar, S. and Schreiner, W. (2005) 'A survey on web services composition Schahram Dustdar * and Wolfgang Schreiner', 1(1), pp. 1–30. Available at: http://www.infosys.tuwien.ac.at/Staff/sd/papers/A_survey_on_web_services_composition_Dustdar_Schreiner_inPress.pdf.
- Endrei, M. et al. (2004) *Front cover Patterns : Service- Oriented Architecture and Web Services ces*. 1st edn. New York: IBM. Available at: <http://>
- Englander, R. (2002) *Java and SOAP, Access*. Sebastopol: O'Reilly. Available at: <http://yag.es/Programming/Oreilly - Java and SOAP.pdf>.
- Galitz, W. O. (2007) *The Essential Guide to An Introduction to GUI Design Principles and Techniques*. 2nd edn, Xtemp01. 2nd edn. New York: John Wiley and Sons.
- Goldman, J. E. (1997) *Local Area Networks: A Client/Server Approach*. 1st edn. New York: Willey and Sons.
- Gould, H. (2008) *Systems Analysis and Design*. 1st edn. Available at: <http://bookboon.com/en/systems-analysis-and-design-ebook>.
- Group, I. S. and Science, C. (2006) 'Guidelines for Requirements Analysis in Students ' Projects', *Information Systems*, pp. 1–32.
- Ince, D. (2004) *Developing Distributed and E-commerce Applications*. 2nd edn. Pearson.
- Kalin, M. (2013) *Java Web Services: Up and Running*. 2nd edn. Beijing: O'Reilly. Available at: www.it-ebooks.info.
- Kotok, A. and Webber, R. R. D. (2002) *ebXML: The New Global Standard for Doing Business on the Internet*. 2nd edn. Indianapolis: Sams.
- Kreger, H. (2001) *Web Services Conceptual Architecture (WSCA 1.0), Architecture*. doi: <http://doi.ieeecomputersociety.org/10.1109/2.982908>.
- Lin, G. C. et al. (2012) 'A Fresh Graduate's Guide to Software Development Tools and Technologies', in, p. 31. Available at: <http://www.comp.nus.edu.sg/~seer/book/2e/Ch10.ServiceOrientedArchitecture.pdf>.
- Natalia, O. and Olifer, V. (2005) *Computer Networks: Principles, Technologies and Protocols for Network Design*. 6th edn. West Sussex: Willey and Sons.
- Newcomer, E. (2002) *Understanding Web Services: XML, WSDL, SOAP, and UDDI*. 1st edn. Addison Wesley.

Noy, N. F. and McGuinness, D. L. (2000) *Ontology Development 101 : A Guide to Creating Your First Ontology*. Available at:
https://protege.stanford.edu/publications/ontology_development/ontology101.pdf.

Ort, E. (2005) *Service-Oriented Architecture and Web Services : Concepts , Technologies , and Tools*. Available at: <http://cic.javerianacali.edu.co/wiki/lib/exe/fetch.php?media=materias:soa2.pdf>.

Peters, J. (2002) *Web Services in the Travel Industry*. Dublin. Available at: <http://www.datalex.com>.

Simão, E. M. (2011) *Comparison of Software Development Methodologies based on the SWEBOK*. Universidade do Minho. Available at:
https://repositorium.sdum.uminho.pt/bitstream/1822/28562/3/eeum_di_dissertacao_pg13264.pdf.

Sommerville, I. (2004) *Software Engineering*. 7th edn. London: Pearson.

Toms, A. (2004) *Threats , Challenges and Emerging Standards in Web Services Security*. Skövde. Available at: <https://www.diva-portal.org/smash/get/diva2:2419/FULLTEXT01.pdf>.