



University of Fort Hare
Together in Excellence

Developing SOA Wrappers for Communication Purposes in Rural Areas

A thesis submitted in fulfillment of the requirements of the degree

Master of Science in Computer Science

at

University of Fort Hare

by

Jimmy Samalenge

November 2010

Acknowledgements

First and foremost I want to thank God, My Heavenly Father and Christ Jesus, My Lord and Saviour. Father, it has been a sweet and sour two years of research, but through it all, You taught me how to put my trust in You. You are the Wind beneath my wings.

To my supervisor Dr Mamello Thinyane: Sir, I want to thank you for your desire to get us to research more and work hard.

To my sponsor Telkom SA: Thank you for helping me further my studies through your financial support.

To my lovely wife and my family: thank you for encouraging me to press forward every time I thought of giving up. Thank you for your prayers and support.

To my classmates, it was nice working together as a family. God bless you all.

Declaration

I, Jimmy Samalenge (Student Number: 200507134), the undersigned acknowledge that all references are accurately recorded and, unless stated otherwise, the work contained in this dissertation is my own original work.

Signature:.....

Date:.....

Publications

Samalenge, J. & Thinyane, M. (2009). Deploying Web Services in Rural Communities for Services of Personal Communication Synchronous and Asynchronous. SATNAC conference, Swaziland.

Samalenge, J., Ngwenya, S., Kunjuzwa, D., Hlungulu, B., Ndlovu, K., Thinyane, M., & Terzoli, A. (2010). Technology Solutions to Strengthen the Integration of Marginalized Communities into the Global Knowledge Society. IST Africa 2010 conference, Durban, South Africa.

Samalenge, J. & Thinyane, M. (2010). Web services communication in a local area network for marginalized communities. ZAWWW 2010 conference, Durban, South Africa.

Acronyms

3GPP	3rd Generation Partnership Project
AMR	Adaptive Multi-Rate
COFISA	Cooperative Framework on Innovation Systems between Finland and South Africa
CSD	Circuit Switched Data
DBMS	Database Management System
DNS	Domain Name Service
EAIF	External Application Interface
FOSS	Free Open Source Software
GIF	Graphics Interchange Format
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HTTP over SSL
IBM	International Business Machines
ICT	Information and Communication Technology
ICT4D	Information and Communication Technology for Development
IK	Indigenous Knowledge
IMAP	Internet Mail Access Protocol
ISP	Internet Service Provider
IM	Instant Messaging
JPEG	Joint Photographic Experts Group
LGPL	Lesser General Public License
LLiSA	Living Labs in Southern Africa
MIDI	Musical Instrument Digital Interface
MIME	Multipurpose Internet Mail Extensions
MMS	Multimedia Message Service
MMSC	Multimedia Message Service Centre
MP3	MPEG Layer-3
MRAs	Marginalised Rural Areas

MTA	Mail Transfer Agent
OMA	Open Mobile Alliance
OS	Operating System
PC	Personal Computer
PNG	Portable Network Graphics
PHP	Hypertext Pre-processor
POP	Post Office Protocol
SIM	Subscriber Identity Module
SLL	Siyakhula Living Lab
SMIL	Synchronized Multimedia Integration Language
SMS	Short Message Service
SMTP	Simple Mail Transfer Protocol
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
TTS	Text To Speech
UDDI	Universal Description, Discovery, and Integration
UI	User Interface
URL	Universal Resource Locator
VAS	Value Added Service
VSAT	Very Small Aperture Terminal
WAP	Wireless Application Protocol
Wi-Fi	Wireless Fidelity
WiMAX	Worldwide Interoperability for Microwave Access
WSDL	Web Service Description Language
W3C	World Wide Web Consortium
WYSIWYG	What You See Is What You Get
XML	eXtensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

Abstract

The introduction of Web Services as a platform upon which applications can communicate has contributed a great deal towards the expansion of World Wide Web technologies. The Internet and computing technologies have been some of the factors that have contributed to the socio-economic improvement of urban and industrial areas. This research focuses on the application of Service-Oriented Architecture (SOA) and Web Services technologies in Information and Communication Technologies for Development (ICT4D) contexts. SOA is a style used to design distributed systems, and Web Services are some of the common realizations of the SOA. Web Services allow the exchange of data between two or more machines in a simple and standardized manner over the network. This has resulted in the augmentation of ways in which individuals in a society and in the world communicate. This research aims to develop a SOA-based system with services that are implemented as Web Services. The system is intended to support communication activities of Dwesa community members. The communication methods identified as the most commonly used in the Dwesa community are Short Message Services (SMSs) and voice calls. In this research we have identified further methods (i.e. Multimedia Message Service, Electronic mail and Instant Messaging) to augment communication activities in Dwesa.

The developed system, therefore, exposes SMS Web Service, MMS Web Service, Email Web Service and IM Web Service that are consumed in machine-to-machine, machine-to-person and person-to-person types of communication. We have also implemented a one-stop communication shop, through a web portal which provides interfaces to the different communication modules.

Elaborate functional and usability testing have also been undertaken to establish the viability and end-user acceptance of the system respectively. This research has provided the initial validation of the effectiveness of the SOA-based system in ICT4D contexts.

Table of Contents

Acknowledgements	i
Declaration.....	ii
Publications	iii
Acronyms	iv
Abstract.....	vi
Table of Figures.....	xii
Table of Listings.....	xiv
1. INTRODUCTION	2
1.1. Introduction	2
1.2. Research Context and Scope	2
1.3. Research problem.....	3
1.4. Research Objectives	3
1.5. Research Methodology.....	4
1.6. Research Deliverables	5
1.7. Structure of Thesis	5
1.8. Conclusion.....	6
2. LITERATURE REVIEW	8
2.1. Introduction	8
2.2. Information and Communication Technologies (ICTs).....	8
2.3. ICT4D.....	9
2.4. ICT4D Technologies	10
2.4.1. Wireless Networks	10
2.4.2. Mobile Phones	11
2.4.3. Social Media	13
2.5. ICT4D methodology: Living Labs.....	14
2.5.1. Definition of a Living Lab	14
2.5.2. Origin of Living Labs	14
2.5.3. Living Labs in Southern Africa (LLiSA).....	15

2.6.	Siyakhula Living Lab (SLL)	15
2.6.1.	SLL location: Dwesa.....	16
2.6.2.	SLL Objectives.	17
2.6.3.	SLL Infrastructure.....	17
2.6.4.	SLL eService applications	19
2.7.	Service-Oriented Architecture (SOA).....	21
2.7.1.	Definition	21
2.7.2.	Origin	21
2.7.3.	SOA Motivation.....	22
2.7.4.	Web Services	24
2.7.5.	History of Web Services	24
2.7.6.	Web Services Standards.....	25
2.7.7.	Web Services Architecture	25
2.7.8.	Web Services Development Phases	26
2.8.	Related Work.....	27
2.9.	Conclusion.....	27
3.	TECHNOLOGIES REVIEW.....	29
3.1.	Introduction	29
3.2.	Hardware Required	29
3.3.	Software Required.....	29
3.3.1.	Kannel	29
3.3.2.	Mbuni.....	31
3.3.3.	Bind9.....	32
3.3.4.	Postfix	32
3.3.5.	Openfire	33
3.4.	Libraries Required.....	33
3.4.1.	NuSOAP	33
3.4.2.	MMSLib.....	33
3.4.3.	XMPPHP.....	34

3.5.	Conclusion.....	34
4.	THE SYSTEM DESIGN	36
4.1.	Introduction	36
4.2.	Functional and non-functional requirements specification	36
4.2.1.	Functional requirements.....	36
4.2.2.	Non-functional requirements	36
4.3.	The Design Scope.....	37
4.4.	The System Architecture	37
4.5.	The User Interface Layer.....	38
4.6.	SOAP-Server and SOAP-Client.....	39
4.7.	SMS Web Service Architecture	40
4.8.	MMS Web Service Architecture	43
4.9.	Email Web Service Architecture.....	45
4.10.	IM Web Service Architecture.....	47
4.11.	System's Users	49
4.12.	Conclusion	49
5.	THE IMPLEMENTATION.....	51
5.1.	Introduction	51
5.2.	SOAP-server.....	51
5.3.	SMS Web Service	54
5.3.1.	The Method.....	54
5.3.2.	The Gateway: Kannel	55
5.3.3.	Connecting to Kannel	56
5.4.	MMS Web Service	58
5.4.1.	The method	58
5.4.2.	The gateway: Mbuni	58
5.4.3.	Connecting to Mbuni	59
5.5.	Email Web Service.....	60
5.5.1.	The method	60
5.5.2.	The DNS	60
5.5.3.	The MTA: Postfix	61

5.5.4.	POP/IMAP Server: Dovecot	62
5.6.	IM Web Service	62
5.6.1.	The Method.....	62
5.6.2.	The XMPP Server	62
5.6.3.	Configuring Openfire and Apache HTTP Server	63
5.6.4.	The IM Client.....	64
5.6.5.	Connecting to the XMPP server	64
5.7.	SOAP-Client.....	65
5.8.	The login interface	68
5.9.	The registration interface	71
5.10.	The welcome page	73
5.11.	The logout page	74
5.12.	SMS Web Service UI	75
5.13.	MMS Web Service UI.....	77
5.14.	EMAIL Web Service UI.....	78
5.15.	IM Web Service UI.....	80
5.16.	Conclusion	81
6.	TESTING AND RESULTS.....	83
6.1.	Introduction	83
6.2.	Testing of back-end Applications	83
6.2.1.	Kannel.....	83
6.2.2.	Mbuni.....	84
6.2.3.	Postfix	84
6.3.	Non-functional testing.....	85
6.3.1.	Compatibility	85
6.3.2.	System Usability Scale.	85
6.4.	Functional Testing.....	90
6.4.1.	Registration	90
6.4.2.	Login.....	91
6.4.3.	Change Password	92

6.4.4. Recover Password.....	93
6.5. Conclusions	94
7. DISCUSSION AND CONCLUSION	96
7.1. Introduction	96
7.2. Achievements	96
7.3. Challenges and limitations	97
7.3.1. Challenges.....	97
7.3.2. Limitations	97
7.4. Future work	97
7.5. Overall conclusion.....	98
8. REFERENCES	100
9. APPENDIX A – Basic Technologies Required	108
10. APPENDIX B - System Implementation Screenshots.....	109
11. APPENDIX C - Configuration files	111
12. APPENDIX D – Code Snippets	119
13. APPENDIX E – Further system testing and results.....	130
14. APPENDIX F – Further usability testing results	131

Table of Figures

Figure 2.1: Mobile subscribes. Mobile supply (Hahn, 2008)	11
Figure 2.2: Communication within Mpume (Pade, et al. 2009)	13
Figure 2.3: Communication with surrounding villages (Pade, et al. 2009)	13
Figure 2.4: SLL Overview	16
Figure 2.5: Dwesa location	17
Figure 2.6: SLL on the ground.....	18
Figure 2.7: A view of point-to-point integration and SOA (Kumar, et al. 2006).....	22
Figure 2.8: System without SOA (Reitman, et al. 2007).....	23
Figure 2.9: System with SOA (Reitman, et al. 2007)	24
Figure 3.1: Kannel as WAP gateway	30
Figure 3.2: Kannel as SMS gateway (Fink, et al. 2010).....	30
Figure 3.3: Mbuni as MMS gateway (Mbuni, 2004).....	32
Figure 4.1: System's overview	38
Figure 4.2: UI Architecture.....	39
Figure 4.3: SMS Web Service architecture.....	41
Figure 4.4: SMS Web Service sequence diagram.....	42
Figure 4.5: MMS Web Service architecture	43
Figure 4.6: MMS Web Service sequence diagram	44
Figure 4.7: Email Web Service Architecture	45
Figure 4.8: Email Web Service sequence diagram	46
Figure 4.9: IM Web Service Architecture.....	47
Figure 4.10: IM Web Service sequence diagram.....	48
Figure 4.11: Use Case diagram.....	49
Figure 5.1: NuSOAP interface.....	53
Figure 5.2: <i>bearerbox</i> and <i>smsbox</i> cron jobs	56
Figure 5.3: Interaction between the UI and SOAP-server via a SOAP-client.	68
Figure 5.4: Login page UI.....	70
Figure 5.5: Recover password UI	70
Figure 5.6: Change password UI	71
Figure 5.7: Validation of the Registration form.....	72
Figure 5.8: Welcome page	73
Figure 5.9: the user has successfully logged out.....	75

Figure 5.10: SMS User Interface	76
Figure 5.11: MMS User Interface	77
Figure 5.12: Email User Interface.....	79
Figure 5.13: IM User Interface	80
Figure 5.14: Link to the IM UI	80
Figure 6.1: <i>bearerbox</i> at work.	83
Figure 6.2: <i>smsbox</i> at work.	83
Figure 6.3: <i>mmsbox</i> at work.....	84
Figure 6.4: User name <i>Jimmy</i> has received the test email.	84
Figure 6.5: A java application communicating with the SOAP-server.	85
Figure 6.6: User's background	86
Figure 6.7: <i>User</i> Registering.....	91
Figure 6.8: Registration of <i>User</i> successful.....	91
Figure 6.9: <i>User</i> logging in.....	92
Figure 6.10: <i>User</i> successfully logged in.....	92
Figure 6.11: <i>User</i> gets a confirmation on the UI.	93
Figure 6.12: Password successfully changed in the users' database.	93
Figure 6.13: <i>User</i> has successfully received a temporary password via email.....	94

Table of Listings

Listing 5.1: Creating a SOAP-server using NuSOAP	51
Listing 5.2: Part of the WSDL document	54
Listing 5.3: <i>smsbox</i> group.....	56
Listing 5.4: <i>sendsms-user</i> group	57
Listing 5.5: <i>mbuni</i> group.....	59
Listing 5.6: <i>send-mms-user</i> group.....	59
Listing 5.7: <i>wapbox</i> group	60
Listing 5.8: <i>db.dwesaproject.com</i> zone file	61
Listing 5.9: <i>main.cf</i>	61
Listing 5.10: Lines 116 and 121 have been uncommented in <i>httpd.conf</i>	63
Listing 5.11: Setting Apache XMPP proxy rule	64
Listing 5.12: Creating the SMS SOAP-client using NuSOAP	66
Listing 5.13: SMS SOAP-Client's call function.	67
Listing 5.14: The <i>auth.php</i> code	69
Listing 5.15: <i>index.php</i>	69
Listing 5.16 <i>logout.php</i>	74

CHAPTER

I

1. INTRODUCTION

1.1. Introduction

This chapter introduces the research presented in this thesis and provides a detailed discussion of the research problem, the key objectives of the research, and the methodology followed to offer a solution to the problem. It also discusses the scope within which the research is done and what is expected to be delivered at the end of the research; before summarising its content, it describes the structure of the thesis.

1.2. Research Context and Scope

This project is undertaken within the context of the Siyakhula Living Lab (SLL), which is based in Dwesa. The SLL is an intervention that explores the use of Information and Communication Technologies (ICTs) for the socio-economic development of marginalized communities through the development of various eServices. Both the SLL and Dwesa are discussed in detail in the second chapter.

As the local market is being replaced by the global market, industries have focussed on becoming increasingly IT flexible in order to keep up with the market's demands. The focus has been on adopting an approach that allows systems to remain scalable and flexible while growing. The approach is known as Service-Oriented Architecture (SOA) (Josuttis, 2007).

SOA has become the architecture that many industries and organizations use to provide support for processes distributed between systems on a local, national and international scale. Departments and business units, in many industries, have started to use the SOA approach to communicate with systems from other departments and/or business units. One of the infrastructures which is used to implement the concept of SOA is Web Services (Josuttis, 2007; Oracle, 2008).

The SOA approach has also been utilized in the SLL context to provide a unified eServices middleware. This allows the Information and Communication Technologies for Development (ICT4D) eServices to be deployed in a distributed, reusable and flexible manner. In view of this evolution of the SLL eServices platform, this research aims to develop a SOA system that enhances and augments communication within the SLL and the Dwesa community.

1.3. Research problem

This research addresses two key problems:

- There are a number of software applications that have been developed within the context of SLL which are almost stand-alone applications. All of these applications interact with the end-user in one way or another. Wertlen states that the effectiveness of these applications can be improved by getting them to collaborate, especially at a contextual level (Wertlen, 2010). As a result, a SOA middleware framework has been deployed in the SLL to create a platform upon which the applications collaborate. While there are core services developed within the middleware (e.g. authentication, profile, data persistence, etc.), it is imperative that other services are developed to support the functioning of the SLL eService platform. Such supporting services include communication between various system modules (e.g. notifications from the network monitoring system). The development of these SOA based communication services is the first problem that this research addresses.
- The SLL exists to provide new technologies and skills to the rural community of the Mbashe municipality, specifically in Dwesa, to improve the quality of life of the people (Pade, et al. 2009). One of the predominant needs within this community is for communication both within the community and also with other distant communities. As such, relevant communication services, consumable by the end users, have to be developed. This is the second problem that the research addresses.

1.4. Research Objectives

The research has two main objectives:

- Objective one: the research aims to deploy a SOA system which implements its functionalities as Web Services for the purpose of enhancing communication at machine-to-machine and machine-to-person levels within the SLL.
 - Machine-to-machine: The focus of this aspect is on the communication between this system's Simple Object Access Protocol (SOAP)-server and other eService applications deployed within SLL. These applications typically require certain services, such as the communication services offered by this project, in their functional requirements. As a result, this project focuses on deploying a SOAP-based

system in such a way that the other eService applications can easily communicate with it and place their requests.

- Machine-to-person: The focus of this aspect is to expose the Web Services in such a way that a machine can consume them for the purpose of communicating with a person. For instance, the eHealth system needs to notify its users, via email, every time the system administrator adds new information in the database. The eHealth system can be programmed in such a way that, whenever new information is added to the database, it automatically sends a pre-set email by requesting the service from the SOAP-server.

In view of this, the research aims to develop SOA wrappers that offer communication services, such as Short Message Service (SMS), Multimedia Message Service (MMS), Electronic mail (Email), and Instant Messaging (IM) which form part of the SOA middleware framework (mentioned under section 1.3) and eliminate the need for each application to produce its own communication service.

- Objective two: the research also aims to provide a person-to-person type of communication. The focus of this aspect is to establish a one-stop-shop for the Dwesa community. This one-stop-shop is a single web-page interface from which users can consume the Web Services for social and business communication.

The following are, therefore, the key sub-objectives of the project:

- Understanding the concept of SOA.
- Identifying key communication requirements within the SLL context.
- Investigating the viability of a SOAP-based Web Services system for communication purposes within the SLL.
- Implementation of the actual system.
- Establishing the usefulness of the system developed.

1.5. Research Methodology

This research methodology is based on the following approach:

- The early stage of the literature review has been based on understanding the concept of SOA and Web Services, especially SOAP-based Web Services.

- Further literature review has been consulted in order to understand how SOAP-based Web Services are implemented. Related work has been thoroughly studied in order to understand the development phases of a SOAP-based Web Service and the technologies needed.
- Trips to the research area have been made in order to study the feasibility of a SOAP-based Web Services system for communication purposes within the SLL.
- Implementation and testing of the system prototype has been done at departmental level in order to assess the usability of the system.

1.6. Research Deliverables

This project has allowed the author to study the concept of SOA as an approach towards developing a scalable and flexible system and having an in-depth understanding of Web Services as one of the foundations of the service-oriented environment.

A system that offers its functionality as Web Services for communication purposes at machine-to-machine, machine-to-person and person-to-person levels, is developed and a web-based interface is added to it to provide a user-friendly user interface (UI).

1.7. Structure of Thesis

The rest of the thesis is structured as follows:

- ***Chapter 2: Literature Review***

This chapter reviews the concept of ICT and distinguishes between old and new ICTs. It also reviews the concept of ICT4D and describes and number of technologies used in ICT4D contexts. It also introduces the research area and the concept of Living Labs. It defines the approach followed in developing the system and discusses related work.

- ***Chapter 3: Technologies Review***

This chapter identifies and discusses the relevant technologies for the implementation of the proposed system.

- ***Chapter 4: The System Design***

This chapter provides a specification of the key system requirements and illustrates the overall design for the complete system consisting of a SOAP-server, SOAP-clients, Web Services and the user interface.

- *Chapter 5: The System Implementation*

This chapter discusses the steps that are followed to implement the entire system.

- *Chapter 6: System Testing and Results*

This chapter discusses the different types of testing the system goes through and the results thereof.

- *Chapter 7: Discussion and Conclusion*

This chapter summarises the work undertaken in this project. It also summarises all successes and challenges in the deployment of the system.

- *References*

- *Appendices*

1.8. Conclusion

This chapter introduces the research study and discusses, in detail, the research problem, the aims and objectives of the project and the structure of the research document. The next chapter discusses the literature consulted during the research process.

CHAPTER

II

2. LITERATURE REVIEW

2.1. Introduction

The previous chapter focuses on introducing the project. This chapter starts by discussing the concept of ICTs and its contribution to rural development. It also considers Dwesa as a representative of rural areas in Africa and goes on to discuss two different approaches towards ICT4D and the concept of living labs. It briefly describes the different Living Labs that exist in Southern African and discusses, in detail, the Siyakhula Living Lab. It continues by reviewing the approach that the system adopts at both its development and implementation stages. Before concluding, the chapter directs its attention to a discussion of a related work and its contribution to this project.

2.2. Information and Communication Technologies (ICTs)

The use of ICTs has made it possible for regional economies, societies and cultures to be integrated into a global network of communication and trade, therefore offering a solution to the distance factor; this is known as globalization (Sayo, 2004:85). They are also very important for sustainable development in developing countries. They have contributed to the changes seen in various aspect of life, such as education, economics, communications, travel and in the way people do business (Thioune, 2003:12). One of the benefits of ICTs is they allow information to be accessed and shared fast and easily, therefore allowing communities to be part of the Knowledge Society.

ICTs are grouped into two categories: the “old” and the “new” ICTs. The “old” ICTs are tools, such as newspapers, televisions and radios, which have been used from many generations ago to disseminate information. These ICTs are cost-effective and do not require any form of qualification or experience to be operated. On the other hand, the “new” ICTs are referred to as new technologies, such as mobile phones, the Internet, personal computers and many other new technologies used for information and communication purposes. The latter are not as cost-effective as the former and they require certain levels of literacy, which does not often come free of charge, in order to operate them. In spite of this, they have greater advantages than the old ICTs and as a result they have been widely used at a socio-economic level; below are the reasons why (Curtain, 2004).

- **Interactivity:** The new ICTs offer the ability for two or more parties to interact with each other using two-way communication.
- **Availability and Accessibility:** The new ICTs, such as the Internet, can be available 24 hours a day making accessibility to information on a real time, synchronous or asynchronous basis.

The new ICTs have permitted communities from various geographical locations to communicate quickly and easily, hence eliminating the issue of geographical distance both in the social and economic sectors.

As the new ICTs are widely used, there has been a progressive reduction in the relative cost of communication although this differs from one region to another (Curtain, 2004).

In spite of the fact that new ICTs offer greater advantages than their older counterparts, the issue of cost and literacy still remains one of the reasons why new ICTs are scarcely used in rural areas. This has led to lack of access to ICTs infrastructure and services therefore creating a digital divide between urban and rural areas (Curtain, 2004).

2.3. ICT4D

ICT4D refers to the use of ICTs within the field of socioeconomic development with the aim of reducing poverty. This is a project that aims to eliminate the digital divide by exposing communities in rural areas to a digital world through the use of ICTs. The lack of ICTs leads the poor to a life which lacks education, income and wealth.

Heeks (2009) explains why the concept of ICT4D is of great importance. The economic, social and political life in the 21st century will become increasingly digital, and those who do not have ICTs will be left behind. In order to have both the rich and the poor move with the digital world, priority should be given to ICT applications for the poor.

There are two approaches to ICT4D (Curtain, 2004):

- **ICT-driven:** under this approach, ICTs are used as tools to promote economic growth. This approach emphasizes that quick and easy ways of communication contribute to economic development. A simple example of this type of approach is the use of eCommerce platforms, in rural areas, to expose business men and women to a larger market.

- ICT-in-support: under this approach, ICTs are used as support systems in on-going projects. The projects already have their goals and objectives set in place, and ICTs are then brought in to help fulfil the goals and objectives in a quicker and more advanced way. An example of this is the use of ICTs in Education.

This project, being an ICT4D project, adopted the first approach. It took advantage of existing ICTs in Dwesa to develop a system that will offer communication services to the Dwesa community.

2.4. ICT4D Technologies

The following are some of the technologies used in ICT4D contexts:

2.4.1. Wireless Networks

Goldsmith (2005) states that a wireless network is commonly associated with a telecommunications network which has interconnections between nodes that are implemented without the use of wires.

WiMAX, Wi-Fi and mobile computer are being used to develop and implement eLearning and wireless technologies in marginalized and rural areas (MRAs). Smyth (2005) states that the use of broadband wireless standards and the implementation of mobile computing can help to:

- Overcome the challenges of terrain, infrastructure, and finance to increase access,
- Deploy broadband quickly and cost-effectively to areas currently not served,
- Extend the benefits of digital education to previously unreachable populations.

The following is a list of wireless network ICT4D projects:

- Uganda's VSAT school-based Telecentre project: This project uses earth-satellite VSAT technology in Uganda to connect schools and communities to the Internet with the aim of creating access to knowledge, educational resources, in order to break isolation and thus foster development opportunities (Schoolnet Uganda, 2007).
- Nepal wireless networking project: The main aim of this project is to bridge the digital divide by means of wireless technology. This project has been able to reduce poverty, create job opportunities, improve communication, encourage eCommerce and increase the quality and availability of healthcare in the rural communities (Pun, et al. 2006).

- Fantsuam Foundation's Community Wireless Network: This project is undertaken in Abuja, Nigeria. Its main purpose is to provide intranet and Internet access to local educational institutions, faith-based institutions, health services, small enterprises and individuals (Flickenger, 2008:309).

2.4.2. Mobile Phones

The demand of mobile phones in Africa has greatly increased over the past few years. Africans are buying mobile phones at a world record rate and this revolution has transformed commerce, healthcare and social lives. Africa is regarded as the fertile market for mobile phones (LaFraniere, 2010).

Figure 2.1: Mobile subscribes. Mobile supply (Hahn, 2008) below shows that the demand of mobile phones in Africa is so significant that it surpasses the actual supply of mobile phones.

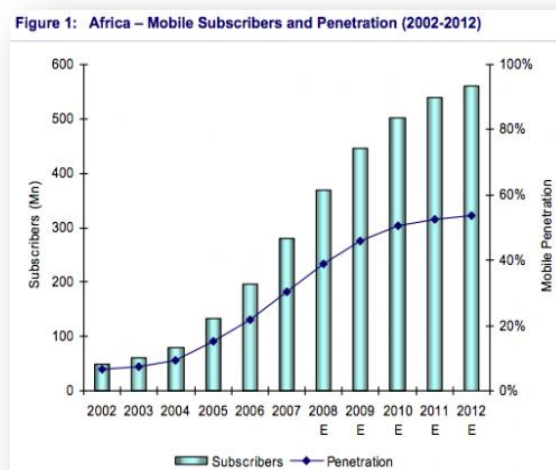


Figure 2.1: Mobile subscribes. Mobile supply (Hahn, 2008)

Traditionally, operators targeted only urban areas, but the demand from rural and low income areas has exceeded their expectations (N. Scott, et al. 2004). Mobile phones are changing the face of the rural world by providing communication and other services at a low cost (LaFraniere, 2010).

The expansion of mobile phones has contributed, to a large degree, to the development of many sectors and aspects of life in Africa.

Burns (2010) has compiled a list of projects that show the contribution of mobile phones in the development of African lives:

- Women and mobile phones: In Egypt, mobile services, such as SMS, are used by women to report sexual harassment.
- Food security: In Uganda, women's groups share mobile phones and radios to participate in agricultural radio shows and communicate with extension workers.
- Health: In Rwanda, the government has deployed mobile phone-based health information systems that provide real-time reporting of field level health data. This allows the government to take fast action.
- Literacy training: In Niger, Literacy training is done through SMS and mobile technology.

In times past, members of MRAs used word of mouth, community meetings, and the writing of notes to communicate with each other (Pade, et al. 2009). The challenge with traditional modes of communication is that they are greatly affected by distance. The further apart communities are from one another the less communication exists between them. Mobile phones have somehow dealt with the distance factor but their level of usage still remains below average. This can be seen in Mpume¹ where only 23% of the population owns mobile phones and only 27% of the population have access to them. Those who own mobile phones often use them when trying to communicate with someone who is considerably far away and, as a result, they are able to reduce their costs (Pade, et al. 2009).

Below are two diagrams that show the use of different modes of communication between members within the same village and between members of a village and its surroundings.

¹Mpume is one of the Villages in Dwesa. Dwesa is described in detail in section 2.6.1

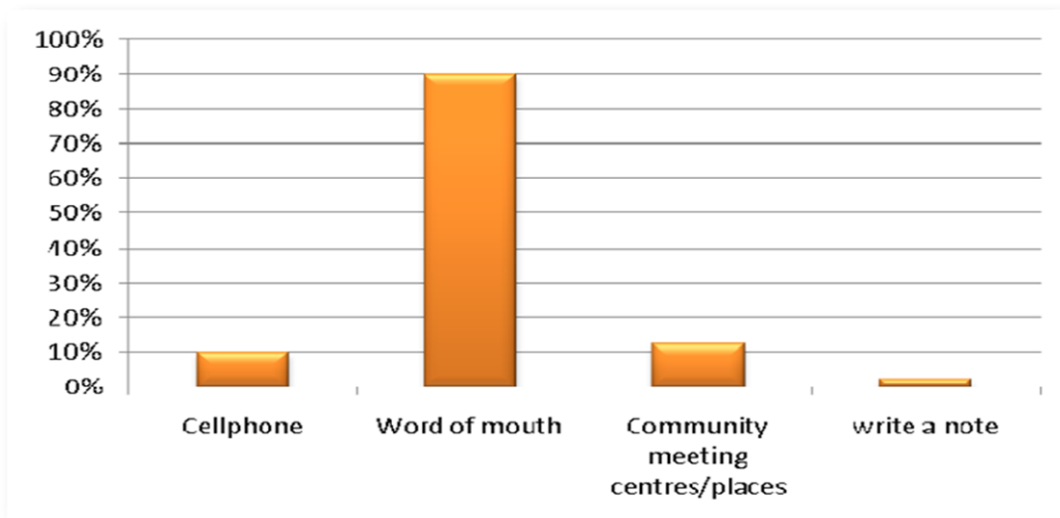


Figure 2.2: Communication within Mpume (Pade, et al. 2009)

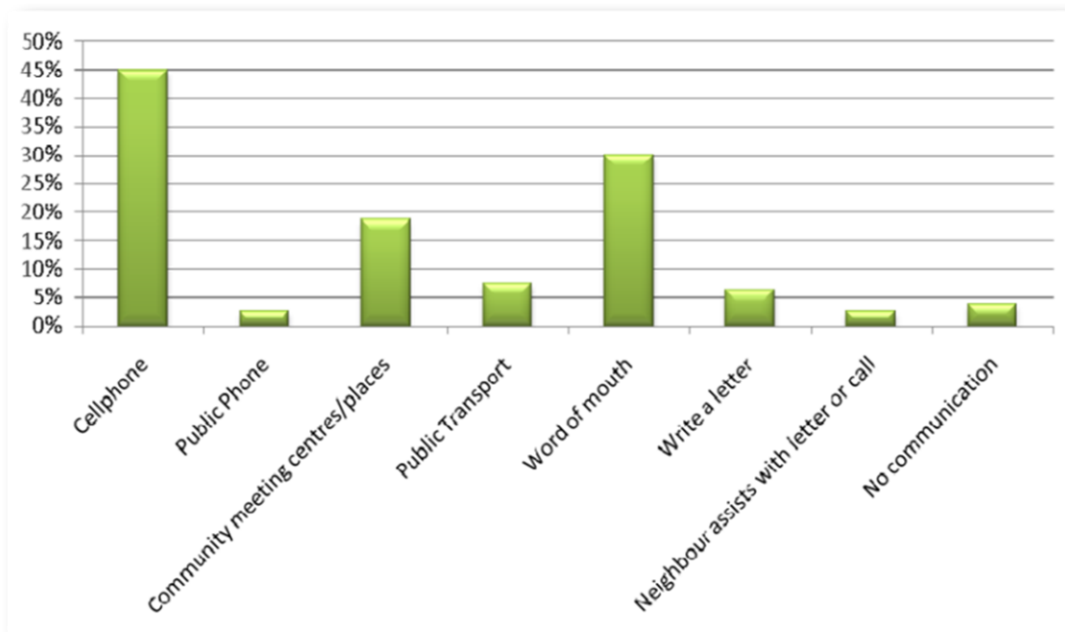


Figure 2.3: Communication with surrounding villages (Pade, et al. 2009)

2.4.3. Social Media

Kaplan and Haenlein (2010) define social media as "a group of Internet-based applications that build on the ideological and technological foundations of Web 2.0, which allows the creation and exchange of user-generated content."

Social media is not only a mere communication channel. It is the widest growing resource since the Internet and the World Wide Web started. It is considered to be something that has changed the way people interact with one another and the way companies interact with

clients. It is something that has changed the ways in which products and services are marketed (Coetsee, 2010).

The following is a list of some popular social media:

- Google
- Facebook
- Yahoo
- Wikipedia
- Twitter

2.5. ICT4D methodology: Living Labs

Living Labs are one of the methodologies used in the ICT4D context.

2.5.1. Definition of a Living Lab

A Living Lab is a user-centred, open innovative ecosystem. It integrates a multidisciplinary research approach and it is community driven. It is also defined as a vehicle designed to put the concept of open-innovation into practice. By taking advantage of pools of creative talents, imagination of the end-user and socio-cultural diversity, a Living Lab enables the development of new services and products.

The concept of Living Labs adopts a triple-helix model of innovation. This model refers to the convergence and crossing-over of the following sectors: public, private and academic (Viale & Ghiglione, 1998). Eriksson, et al (2006) state that innovation generated from two out of the three above sectors would have serious constraints. The triple-helix model has also, in the SLL context, been adapted and referred to as the quadruple-helix model, to emphasize the centrality of the end-users.

2.5.2. Origin of Living Labs

The concept of Living Labs was developed by Professor William J. Mitchell of the MIT Media Lab and School of Architecture. He proposed user-centric research methods that will help keep parties involved up to date with the continuously growing society and work environment. The focus of his research methods is to identify and build prototypes, and to evaluate multiple solutions (Helsinki Living Lab, 2007).

2.5.3. Living Labs in Southern Africa (LLiSA)

LLiSA is a project that focuses at building a network and community of Living Labs practitioners in Southern Africa, with the purpose of advancing and supporting open user-centric innovations and Living Labs in South Africa (Living Labs in Southern Africa, 2010).

There are a number of Living Labs that are part of the LLiSA (Living Labs in Southern Africa, 2010):

- Athlone Living Lab
- Bushbuck Ridge Living Lab
- Limpopo Living Lab
- Moutse Living Lab
- Soshanguve Living Lab
- Sekhukhune Living Lab

2.6. Siyakhula Living Lab (SLL)

The SLL was initiated in 2005 in partnership between the Telkom Centre of Excellence at Fort Hare University and Rhodes University. In 2008, it was formalized into the Siyakhula Living Lab through the financial support of the Cooperative Framework on Innovation Systems between Finland and South Africa (COFISA).

This project is a multi-stakeholder, multidisciplinary intervention consisting of academia, industry, government and marginalized communities. The project facilitates user-driven innovation in the ICT4D domain through the support of other entities within its ecosystem (Dalvit et al, 2007).

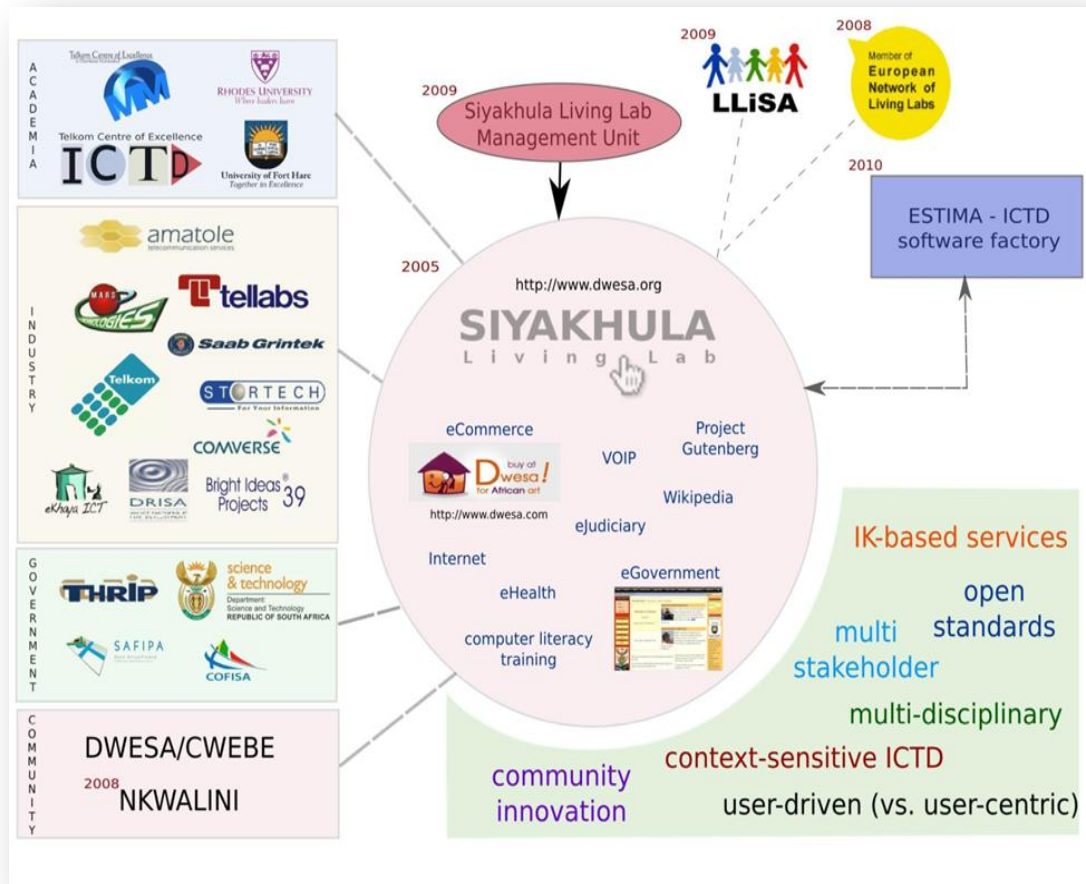


Figure 2.4: SLL Overview

2.6.1. SLL location: Dwesa

Dwesa is a rural area that is, in many ways, a representative of rural areas in South Africa and in the rest of the African continent.

Dwesa is located in the Wild-Coast, in Transkei, in the Eastern Cape province of South Africa. The Wild Coast is situated far from large cities, consequently far from tarred roads and airports. It recently saw the arrival of piped water and the area is currently undergoing the installation of electrical poles and electrical connections into homes.

Dwesa has five villages, namely Mpume, Ngwane, Nqabara, Ntokwane and Nondobo, and its population is currently 15000 people who live in, roughly, 2000 households. The locals have no experience in industrial services because this is an area of subsistence farming and migrant labour and they are also heavily dependent on government subsidies (Timmermans, 2004).

The presence of a nature reserve and the rich cultural heritage of Dwesa give the region great eco-cultural tourism potential. There are two major projects undertaken in Dwesa that aim to contribute towards its socio-economic improvement, namely the marine conservation project undertaken at the nature reserve and the Siyakhula Living Lab project which is described under section 2.6 of this document.

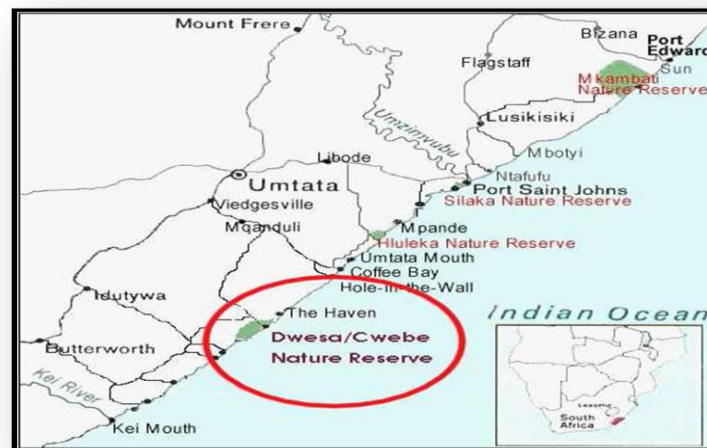


Figure 2.5: Dwesa location

2.6.2. SLL Objectives.

The following are the objectives of the SLL (Dalvit et al, 2007):

- To develop innovative user-driven context-sensitive eServices for the direct benefit of the communities.
- To distribute community telecommunication infrastructure.
- To provide community empowerment through computer literacy training.
- To network the community and build the bridge to a knowledge society.

2.6.3. SLL Infrastructure

The following image depicts the Local Area Network (LAN) deployed in Dwesa.

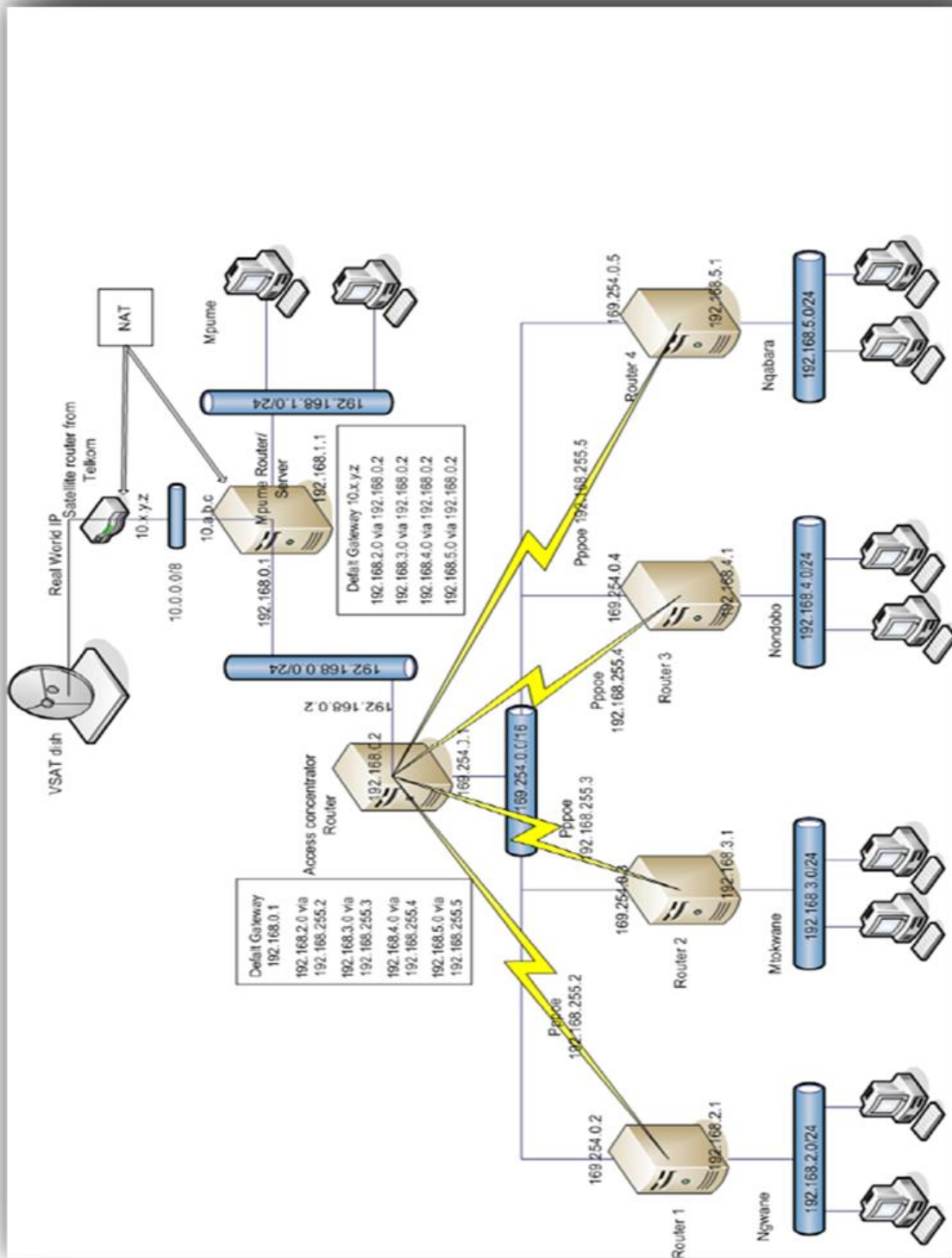


Figure 2.6: SLL on the ground

2.6.4. SLL eService applications

There are a number of projects under the SLL umbrella. Most of them are web-based eServices developed in the standard client/server environment.

The following is a list of web-based eServices applications already deployed:

- eCommerce - this service was developed to expand the financial revenue spectrum by exposing local business products to the rest of the world (since they only depended on local revenues) and, in the process, to expose the same business people to the use of ICTs for business purposes (Dalvit, et al. 2007).
- eGovernment - Dwesa is located 47 km outside of Willowvale. In order for the community of Dwesa to access basic public services (e.g. applying for an identity document), they have to travel the distance. Disadvantaged rural communities in South Africa have suffered from little to no public services provision. The eGovernment system was developed to remedy such issues (Jakachira, 2009).
- eJudiciary – Dwesa legal issues have been always been discussed at the chief’s house under a tree, and once the issues have been resolved, the people are dismissed. A few months later, a little is remembered concerning the matter. The eJudiciary services augments traditional judiciary processes by making available a way of safeguarding vital legal data, and guaranteed persistence and availability of data. It also makes legal information easily available to the public (M. Scott, 2010).

The following is the list of eServices applications that are still being developed:

- Investigating next-generation services architectures for ICTD contexts: This project investigates the current trends that will shape the future architectures of web applications. Based on the projection of the technology, the project seeks to develop frameworks that will enable the seamless migration and integration of these new developments in service platforms for marginalized rural areas.
- Developing a help-desk system for a multi-purpose ICT platform in a marginalized setting, Dwesa (A case study): This project aims to enable technical sustainability and up-skilling in the community.

- Novel interaction techniques for mobile phones: This project aims to investigate novel interaction techniques for mobile applications developed in the ICTD context.
- Cost-sharing and revenue management system for Siyakhula Living Lab: The project investigates financial sustainability in Dwesa, and explores the implementation of context-sensitive billing matrices for ICTD interventions.
- Developing eServices adaptors for the SOA middleware: This project aims to develop Web Services adaptors for eServices that have been developed as standalone web applications previously in the Siyakhula Living Lab. This is to enable these services to be accessible through the SOA middleware that is being developed as the core of the SLL eservices platform.
- Building an e-health component for a multipurpose communication centre: This project aims to develop an eHealth portal for Dwesa, using the latest tools available for the semantic web. The portal will also facilitate the collection and codification of local medical Indigenous Knowledge.
- Experimentation with Mobile WiMAX in ICTD contexts: This project will investigate the viability and effectiveness of Mobile WiMAX for network service delivery in ICTD contexts.
- Developing Xhosa audio/telephony tools for ICTD contexts: This project will build Text To Speech and Automatic Speech Recognition tools for the Xhosa language, to be used within the Siyakhula Living Lab for the provision of telephony based services.
- Exploring user-driven telephony services in an ICTD context: This project aims to develop a platform to facilitate user-driven orchestration of telephony services
- Deployment and extension of a converged WiMAX/WiFi network for Dwesa: This project aims to deploy further WiMAX points on the Siyakhula Living Lab network, and

to setup WiFi hotspots around the schools. The investigation of the effectiveness of the converged WiMAX / WiFi network for the ICTD context will be undertaken.

All the projects listed above collaborate through the SOA middleware framework mentioned in section 1.3. The next section introduces the concept of SOA.

2.7. Service-Oriented Architecture (SOA)

This section discusses the SOA and how it can be implemented in systems development.

2.7.1. Definition

SOA has no fixed definition. The World Wide Web Consortium (W3C) refers to SOA as a set of components which can be invoked, and whose interface descriptions can be published and discovered. According to Wikipedia, SOA expresses a software architectural concept that defines the use of services to support users' requirements.

Two terms need to be considered when it comes to SOA (Srinivasan & Treadwell, 2005).

- **Service:** This is a software component that offers functionality to a service requester via the network.
- **Architecture:** This refers to the style or manner in which the system is designed; it describes the system's purpose, functions, externally visible properties and interfaces.

After defining the two terms, SOA can be defined as a style of designing a reliable distributed system that provides functionality as services. SOA refers only to the design of the system and not its implementation.

Unlike the traditional point-to-point architectures, SOA comprises of loosely-coupled and highly interoperable application services.

2.7.2. Origin

Josuttis (2007:7) states that 'Alexander Pasik, a former analyst at Gartner, coined the term SOA for a class on middleware that he was teaching in 1994. Pasik was working before eXtensive Markup Language or Web Services were invented, but the basic SOA principles have not changed. Pasik was driven to create the term SOA because "client/server" had lost its classical meaning. Many in the industry had begun to use "client/server" to mean distributed computing involving a PC and another computer. A desktop "client" PC typically

ran user-facing presentation logic, and most of the business logic. The back-end “server” computer ran the database management system, stored the data, and sometimes ran some business logic. In this usage, “client” and “server” generally referred to the hardware. The software on the front-end PC sometimes related to the server software in the original sense of client/server, but that was largely irrelevant. To avoid confusion between the new and old meanings of client/server, Pasik stressed “server orientation” as he encouraged developers to design SOA business applications.

2.7.3. SOA Motivation

Unlike the point-to-point integration which forces the service provider to attend to clients’ needs separately, which is time consuming and offers little reuse of the integration interface, SOA provides an environment within which providers design and offer their products as reusable services. If the products are correctly created, not only will they provide services and support to current users, but also to unanticipated ones resulting in providers not having to deal with users as separate entities (Kumar, et al. 2006).

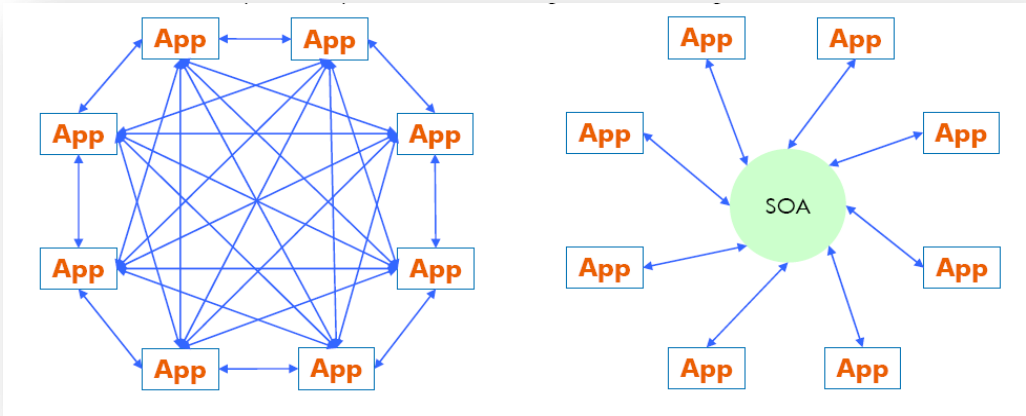


Figure 2.7: A view of point-to-point integration and SOA (Kumar, et al. 2006).

The SOA approach emphasises loose coupling between interacting services (Josuttis, 2007; Srinivasan & Treadwell, 2005).

The loose coupling concept focuses on reducing system dependencies which in turn reduce system failures. Under this concept, the following can be achieved (Srinivasan & Treadwell, 2005):

- **Reusability:** Loosely coupled services can be reused. The concept of reusability saves a lot of time at the development stage, because developers can take already existing services and reuse them to meet new business requirements.
- **Interoperability:** One of the objectives of the SOA approach is for the client and services to communicate with each other in such a way that they understand each other regardless of platforms, systems and languages.
- **Scalability:** Services can be removed or added as demand varies.
- **Flexibility:** Unlike tightly-coupled services which share semantics, states and libraries and are difficult to constantly modify, loosely-coupled services are more flexible because they can be easily updated and modified to keep up with the constantly growing market needs and requirements.
- **Cost Efficiency:** The fact that loosely-coupled services are reusable and flexible makes them less costly.

Below, Figure 2.8 shows a system that runs three processes which have a login/Authentication service duplicated within each of them. This is the approach the SLL followed when developing current eServices (eCommerce, eGovernment and eJudiciary). Directly below it, Figure 2.9 shows the same system but, this time, migrated to SOA. The login/Authentication service is no longer part of each process, but it has become a separate service which allows the user to login once and have access to each process.

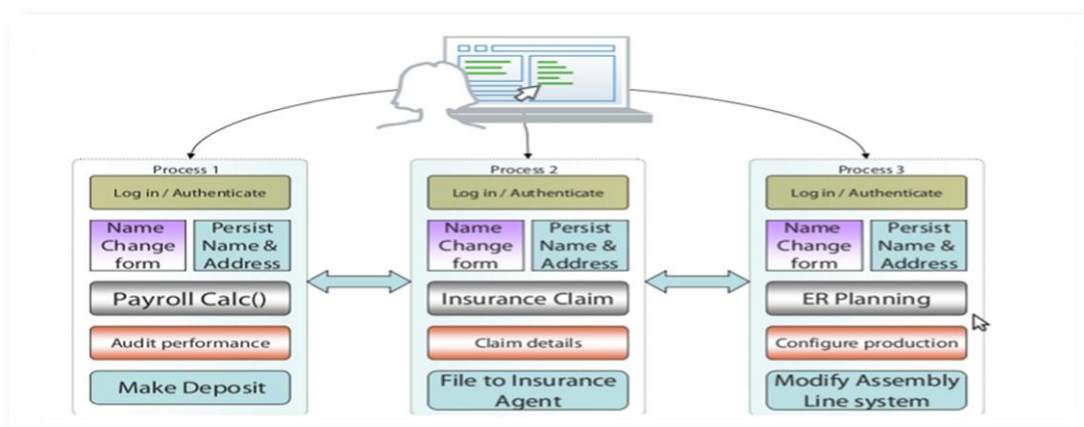


Figure 2.8: System without SOA (Reitman, et al. 2007)

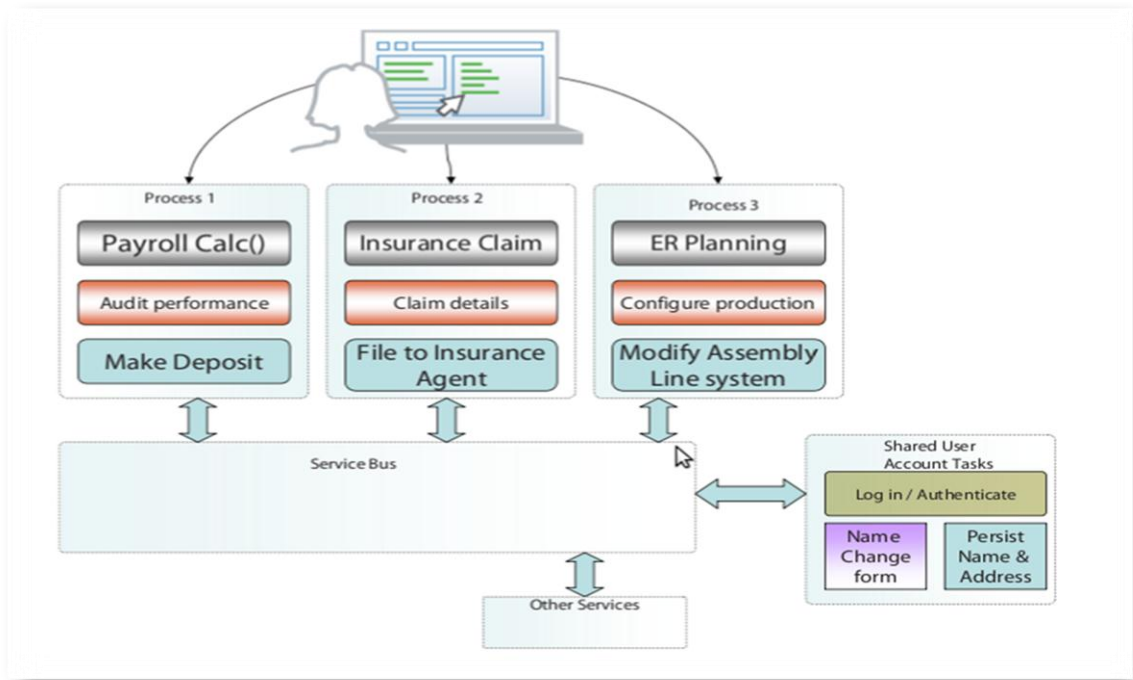


Figure 2.9: System with SOA (Reitman, et al. 2007)

2.7.4. Web Services

Srinivasan and Treadwell (2005) state that Web Services are well suited as the basis of a service oriented environment. Josuttis (2007:209) says that ‘Web Services are widely regarded as the way SOA should be viewed in reality’. They are not mutually dependent because SOA is an architectural style while Web Services are an implementation technology.

According to Richards (2006:11), Web Services do not have a single definition. He states that a Web Service can be defined as a software system that is designed to support machine-to-machine interoperability over a network. A Web Service has an interface written Web Service Description Language (WSDL), and it communicates with other systems using SOAP messages in a way defined by its description. The communication is done over a native network protocol such as HyperText Transfer Protocol (HTTP) in conjunction with many other web-based standards (Richards, 2006).

2.7.5. History of Web Services

In the year 2000, Microsoft coined the term Web Services to describe a set of standards to permit computer-to-computer communication via a network, which includes the Internet (Josuttis, 2007:210).

Initially, Microsoft, and others, used SOAP as a protocol that uses eXtensible Markup Language (XML) to transfer data over a connection such as HTTP and Transmission Control Protocol/Internet Protocol (TCP/IP). Later in the same year, IBM joined Microsoft and others; as a result the WSDL and the Universal Description, Discovery and Innovation (UDDI) standards were brought into the picture. Oracle, HP and Sun announced the support and deployment of Web Services standards in their products by the end of the year 2000 (Josuttis, 2007:210; Richards, 2006:10).

At present, there are more than 50 Web Services standards and 10 profiles, specified by different standard bodies such as the W3C, OASIS, and the Web Services Interoperability Organization (WS-I), which cover almost all areas of distributed computing and remote service calls (Josuttis, 2007:210).

2.7.6. Web Services Standards

Web Services use five fundamental standards. Two of them existed before web Services (Josuttis, 2007):

- XML: This element is used to code and decode data. It is used to describe models, format and data type. Web Services standards are based on XML schema definition and XML namespace.
- HTTP (This includes also HTTPS): This is a low-level protocol used by the Internet. It is also used to access Web Services over the network using Internet technologies.

The following three are specific to Web Services:

- SOAP: This is the protocol that specifies how structured information implemented in Web Services is exchanged in computer networks.
- WSDL: This is an XML document that describes the type of service available in the Web Service and gives a detailed description of the actual Web Service.
- UDDI: This provides a mechanism for the client to find Web Services.

2.7.7. Web Services Architecture

Web services architecture comprises of three roles, which interact with each other. These three roles involve three operations and they are all based on the Web Services artefacts (Josuttis, 2007).

- Roles: Services provider, service requestor and service registry.
 - The service provider: This can be the service owner or the platform upon which the service is hosted.
 - The service requestor: This can be a person requesting the service via a browser or other services requesting the service through a back-end operation.
 - The service registry: This is a registry in which service providers publish their service descriptions. This registry can be searched by service requestors.
- Operations: Publish, find and bind.
 - Publish: A service provider needs to publish a service description in order for it to be accessible.
 - Find: Under this operation, the service requestor either retrieves the service description directly or queries the registry.
 - Bind: After the service requestor has collected the service descriptions, under the bind operation, the service is finally invoked using the binding details from the service descriptions.
- Web Services artefacts
 - Services: A service is a software module deployed on the network by the service provider and it is invoked by the service requestor. A service is the implementation of a Web Service where the interface is the service description.
 - Service descriptions: These are published to a service requestor or to a registry and they contain data type, operations, binding information and network location, to say the basics.

2.7.8. Web Services Development Phases

There are four phases in the development lifecycle of Web Services (Josuttis, 2007).

- Build: This phase includes the following
 - The development and testing of Web Services implementation.
 - The definition of the service interface description.
 - The definition of the service implementation description.
- Deploy: At this stage, executables for the Web Service are deployed into an executing environment and the service descriptions are published to the service requestor or service registry.
- Run: At this stage, the service is fully deployed and available to the service requestor

- Manage: Finally, the availability, quality of service, performance and security of the Web Service is constantly managed and administered.

2.8. Related Work

The Alcatel XML Web Services project is closely related to this research. It is a suite of powerful services that Alcatel offers to developers to benefit from its communication solutions (Alcatel-Lucent, 2006).

This suite offers the following services (Alcatel-Lucent, 2006):

- My Phone Web Services: They allow an external application to handle a user's calls on his/her phone set and to configure a subset of parameters of the phone set. Those parameters define the behavior of the phone set.
- My Messaging Web Services: They are used to interact with an Alcatel Voicemail 4635 or an Alcatel Voice Mail 4645.
- My Assistant Web Services: They give an external application the ability to set and review the rules that route the incoming calls of a user.
- My Management Web Services: They allow the configuration of a subset of parameters of a user's phone set. Those parameters are modified when assigning a phone set to an employee.

Though the services offered, in the above project, are not identical to those of this research, the concept, style and use of technologies are closely related.

2.9. Conclusion

This chapter discusses the literature that was consulted prior to the system's development. It defines, in detail, the proposed approach of the system's development and how to implement this approach. It briefly discusses a related project before its conclusion. The next chapter focuses on the various technologies required in the development of the system.

CHAPTER III

3. TECHNOLOGIES REVIEW

3.1. Introduction

The previous chapter focused on the researcher's literature review. This chapter focuses on the technologies needed to develop a SOAP-based Web Services system.

3.2. Hardware Required

A GSM/GPRS modem is needed since the system involves sending of SMSs and MMSs. A subscriber identity module (SIM) card is needed in order to establish connection with the existing SMS and MMS centres.

More information on hardware required is found in Appendix A.

3.3. Software Required

The choice of software required for the development of this system is limited to Free and Open Source Software (FOSS) because of the type of platform upon which the system is deployed. SLL projects are developed upon and with FOSS only.

The type of Operating System (OS) upon which the system is developed, the web server, the database management system and the scripting language used to develop the system are described in Appendix A.

3.3.1. Kannel

Kannel is a Wireless Application Protocol (WAP) and Short Message Service (SMS) gateway.

As a WAP gateway, Kannel is used for connecting WAP phones to the Internet. WAP is a collection of various languages and tools and an infrastructure for implementing services for mobile phones. WAP makes it possible to implement services similar to the WWW. A WAP gateway is a gateway between phones and servers through which WAP provides contents to phones using WAP protocol stack and translates the requests it receives to normal HTTP (Fink, et al. 2010).

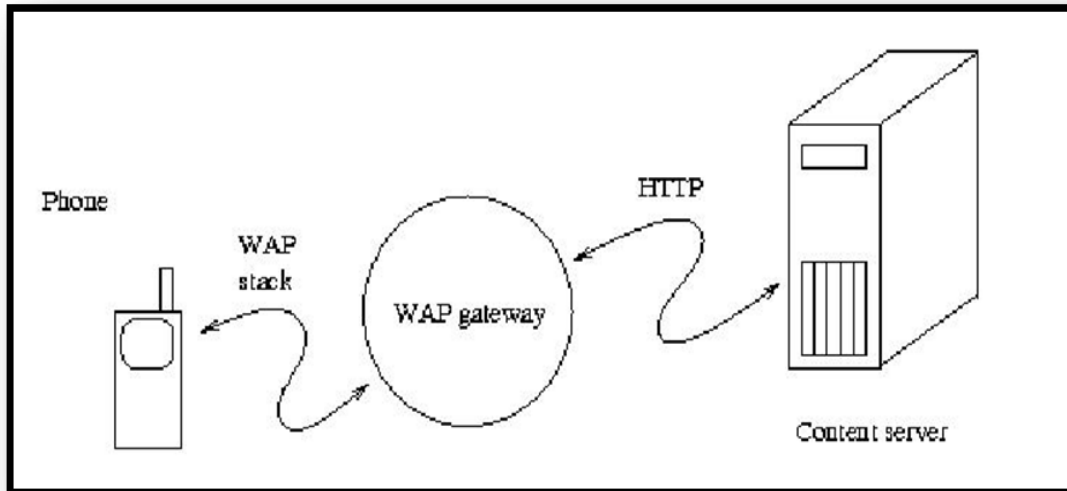


Figure 3.1: Kannel as WAP gateway

As a SMS gateway, Kannel is used to handle connections with SMS centres (SMSCs) and to relay them onward in a unified manner. SMS is a way to send short (160 character) messages from one Global System for Mobile (GSM) phone to another. It can also be used to send regular text as well as advanced content like operator logos, ringing tones, business cards and phone configurations (Fink, et al. 2010).

SMS services are content services initiated by an SMS message to a certain phone number, which then answers with requested content, if available.

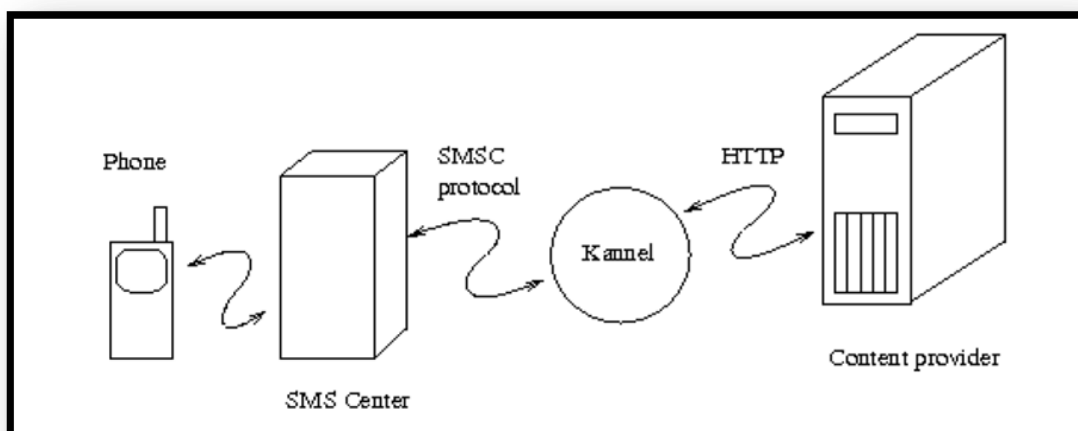


Figure 3.2: Kannel as SMS gateway (Fink, et al. 2010)

3.3.2. Mbuni

This is a Multimedia Message Service (MMS) gateway which implements all major MMS interfaces such as phone-to-phone (MM1 interface), phone-to-email (MM3), inter-MMSC (MM4) and MMS Value Added Service (VAS) (MM7) (Mbuni, 2004).

Mbuni includes a fully-fledged MMS switching centre (MMSC) for the network operator and it also includes a MMS VAS gateway, known as the *mmsbox*, for the MMS content provider. Mbuni is designed to work dependently of Kannel. Mbuni uses Kannel's *smsbox* to send a SMS notification, whenever a MMS has been sent, to the destination number (Mbuni, 2004).

Mbuni supports current generation multimedia messaging and it can be operated in two different modes (Mbuni, 2004):

- As a MMSC it provides the following:
 - Phone-to-phone messaging.
 - Automatic content adaptation: The server modifies message content depending on the capabilities of the receiving terminal.
 - Integrated Email-to-MMS and MMS-to-Email gateway.
 - Support for persistent storage of messages for subscribers.
 - Inter-MMSC message exchange (MM4 interface).
 - Support for MMS VAS Providers using MM7 protocols (SOAP or EAIF).
 - Support for integration with subscriber database to enable smart handling of handsets that do not support MMS, handsets not provisioned, etc.
 - Support for flexible billing structure through billing/CDR plug-in architecture.
 - Bearer (data) technology neutral: Works with GSM/CSD or GPRS.
- As VAS Gateway it provides the following:
 - Support for both SOAP and External Application Interface (EAIF) connectivity with an operator MMSC.
 - Multiple connections to different MMSC of different types can be maintained.
 - MMS content can be loaded from file, URL or as the output of a program.
 - MM composition from Synchronized Multimedia Integration Language (SMIL): The gateway will automatically fetch all components referenced in the SMIL and add them to the MM.
 - A URL interface for MM dispatch.

Mbuni is designed and tested to conform to Open Mobile Alliance (OMA), WAP and 3rd Generation Partnership Project (3GPP) MMS standard. Just like Kannel, Mbuni can be easily customized to suit its user's requirements (Mbuni, 2004).

MMS is intended to provide users with a rich content format including pictures, audio and games. Unlike SMS, which is transported over a traditional GSM network, MMS is designed to be transported over WAP and is encoded using WAP Multipurpose Internet Mail Extensions (MIME) format (Mbuni, 2004).

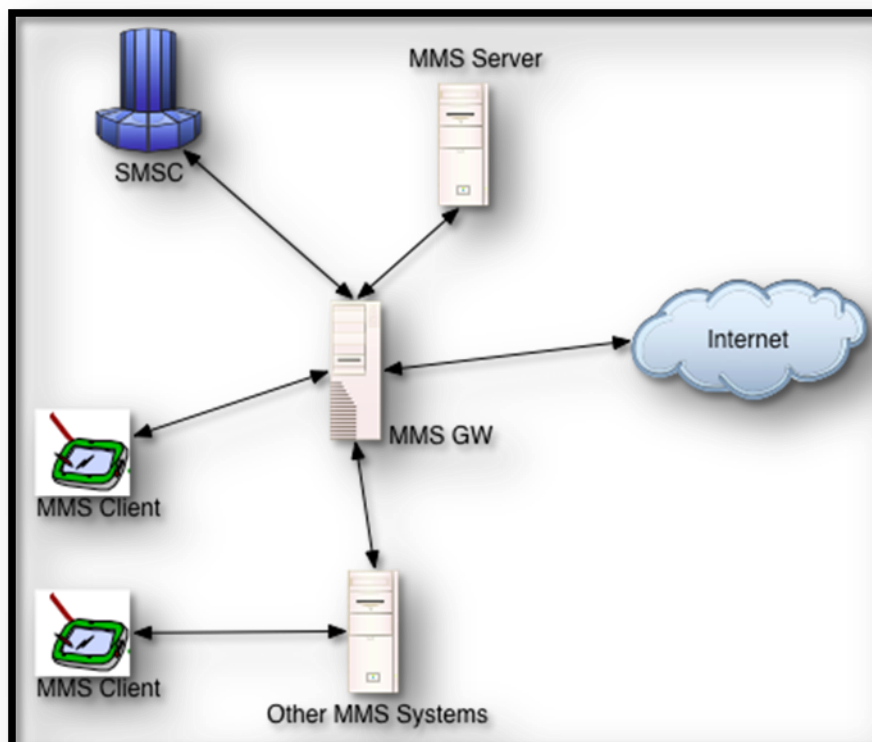


Figure 3.3: Mbuni as MMS gateway (Mbuni, 2004)

3.3.3. Bind9

Bind9 is a Domain Name Service (DNS). A DNS is used for name resolution. Humans work better with names and computers work better with numbers. So a DNS translates IP addresses (which are numbers that computers use to talk to each other over the Internet) into DNS names and vice versa (Vugt, 2008:255).

3.3.4. Postfix

Postfix is a mail transfer agent (MTA) that routes and delivers electronic mail.

3.3.5. Openfire

Openfire is a XMPP server, written in java, which will be incorporated into the system for the purpose of offering IM services to users.

Openfire is platform independent and contains the following features:

- User-friendly web interface and guided installation
- Web-based administration panel
- Plug-in interface
- Customizable
- Database connectivity
- Storing messages and user details

3.4. Libraries Required

This section discusses the various libraries required in order to develop some components of the system.

3.4.1. NuSOAP

NuSOAP: NuSOAP is a rewrite of SOAPx4, provided by NuSphere. NuSOAP is a group of PHP classes that allow developers to create and consume SOAP web services. NuSOAP does not require any special PHP extensions, which makes it usable by all PHP developers, regardless of ISP, server or platform. NuSOAP is licensed under the LGPL (Crugnalo, 2006).

NuSOAP is the technology used in this project to write both the SOAP-server and the SOAP-clients. Initially, SOAPx4 library was used to write both the SOAP-client, the SOAP-server and consume SOAP Web Services, but the only problem with this is that the developer had to write the whole WSDL document, line by line, which is time consuming and may contain errors (Richards, 2006:723). With NuSOAP, the WSDL document is automatically created when creating the SOAP-server (This is discussed in detail in section 5.2).

3.4.2. MMSLib

This is a PHP library that enables encoding and decoding of MMSs. With this library, one can create messages and add multimedia parts such as Joint Photographic Experts Group (JPEG), Graphics Interchange Format (GIF), Adaptive Multi-Rate (AMR) and Musical Instrument

Digital Interface (MIDI). The library also contains a MMS functionality that uses HTTP service to send SMS notifications (Craydon, 2004).

3.4.3. XMPPHP

XMPPHP is the most widely used PHP jabber library for sending and receiving chat messages (Wassermann, 2009).

3.5. Conclusion

This chapter reviews the different technologies that are needed to develop and implement the system. The next chapter discusses the architecture of the system.

CHAPTER

IV

4. THE SYSTEM DESIGN

4.1. Introduction

The previous chapter describes the technologies required for the development. This chapter identifies the key requirements for the system and discusses the conceptual architecture of the system. It defines the scope within which the system is designed and implemented and elaborates on each component of the system.

4.2. Functional and non-functional requirements specification

This section discusses the functional and non-functional requirements specification. This is the requirements specification the system is tested against in order to verify whether or not the objectives are met.

4.2.1. Functional requirements

The following are the key functional requirement that have been identified:

- Provide SOA interface to SMS, MMS, Email + IM.
- Provide machine-to-machine connectivity: for example, the ability of the eCommerce application to connect to the SOAP-server for utilizing any of the communication channels.
- Provide machine-to-person interaction: for example, the ability of the network monitoring system to send an SMS to its administrator whenever the network is down.
- Provide a UI that allows the user to do the following:
 - Create an account
 - Login
 - Change password
 - Recover password
 - Consume Web Services

4.2.2. Non-functional requirements

The two key non-functional requirements that have been identified are as follows:

- Compatibility

- Usability
 - effectiveness (can users successfully achieve their objectives)
 - efficiency (how much effort is expended in achieving those objectives)
 - satisfaction (was the experience satisfactory)

4.3. The Design Scope

The advent of new technologies, such as the Internet and mobile phones, has reshaped the way people communicate with one another. Internet technologies, such as email, social media and IM, have created a qualitatively different communication medium (Iskold, 2007). As discussed under section 2.4.2, mobile phones are changing the face of the rural world because of their low cost services.

This project is designed to take advantage of the above mentioned communication technologies in order to provide the SLL and the Dwesa community with a modern type of communication medium.

The following points have been considered to define the design scope:

- Due to the nature of the platform upon which this project is deployed and human resource constraints, this project's implementation is limited to the use of FOSS.
- The existence of more than one network coverage (Vodacom and MTN), in Dwesa, makes it possible to choose the most reliable network.
- The existence of Living Labs in five schools, which are wirelessly connected, sets a good environment for the deployment of the system.

4.4. The System Architecture

The term 'system architecture' is defined as the overall design and structure of a system or computer network.

This section defines the manner in which the entire system is structured and it describes the different components needed for a fully functional system.

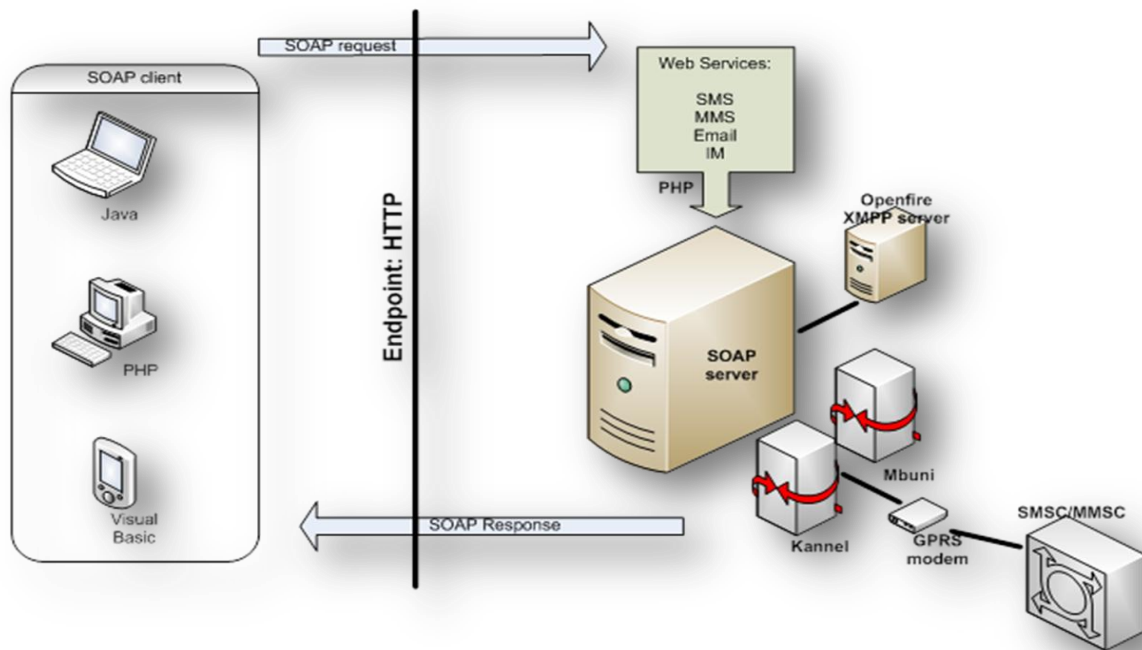


Figure 4.1: System's overview

4.5. The User Interface Layer

This system is designed in a client-server environment and to meet its second objective, it needs a UI. In order to realize a person-to-person type of communication, a user interface is needed. This allows the user to interact with the system and consume the services available in an easy and understandable way.

The UI layer provides interfaces that the user interacts with in order to make use of the services the system offers. The layer consists of pages and the relationship between them. The diagram below highlights the UI model that this layer follows.

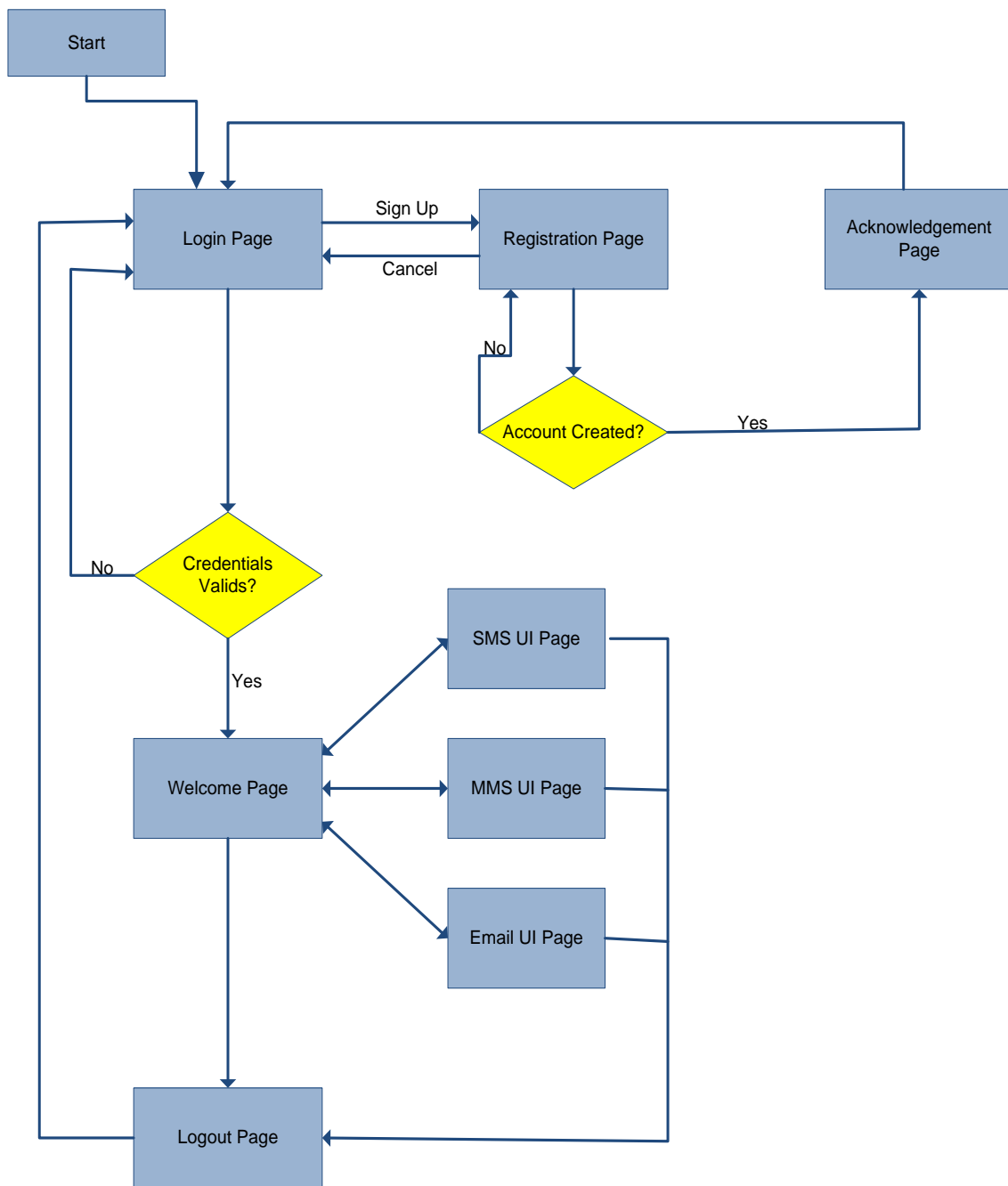


Figure 4.2: UI Architecture

4.6. SOAP-Server and SOAP-Client

This system offers users SOAP-based Web Services for communication purposes. Before discussing each Web Service, there is a need to first understand the SOAP-client and the SOAP-server and how they work together. As stated in section 3.4.1, both the SOAP client and server are created using NuSOAP.

Richards (2006:710) defines the *SOAPClient* class as the pillar for consuming SOAP-based Web Services. It can make requests to a SOAP-server and can also perform tasks such as data conversion and encoding that are needed in order to create SOAP messages using PHP variables and types. He also defines a SOAP-server as the house of Web Services. It is based on a WSDL document which the SOAP client makes use of in order to get detailed descriptions of Web Services offered. The WSDL document describes how a Web Service is accessed, what it is designed to do and how messages are passed.

The SOAP-client and SOAP-server communicate by sending and receiving SOAP messages in a request and response format. This SOAP request-response process is done using `HTTP_RAW_POST_DATA` (Richards 2006:617). This function is defined in the SOAP-server.

4.7. SMS Web Service Architecture

Bodic (2005:47) defines the SMS as being a basic service allowing the exchange of short text messages between subscribers.

The SMS Web Service application allows a person-to-person and a machine-to-person type of communication in the form of SMSs. This application uses a PHP function, registered as a method in the SOAP-server, which connects the system to the SMS gateway. The method receives a call from the SOAP-client, requesting to send an SMS to a specified destination, and execute it by sending the request to the gateway. The gateway sends the SMS to the recipient's number via a modem which is connected to a SMSC.

This is illustrated in the two diagrams below. Figure 4.3 shows the different components involved in sending a SMS and Figure 4.4 shows a chain of requests and responses involved in sending a SMS.

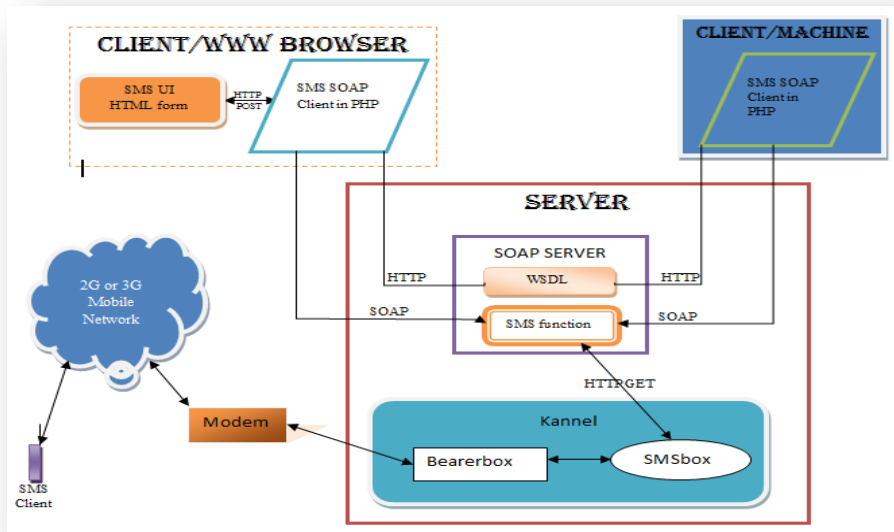


Figure 4.3: SMS Web Service architecture

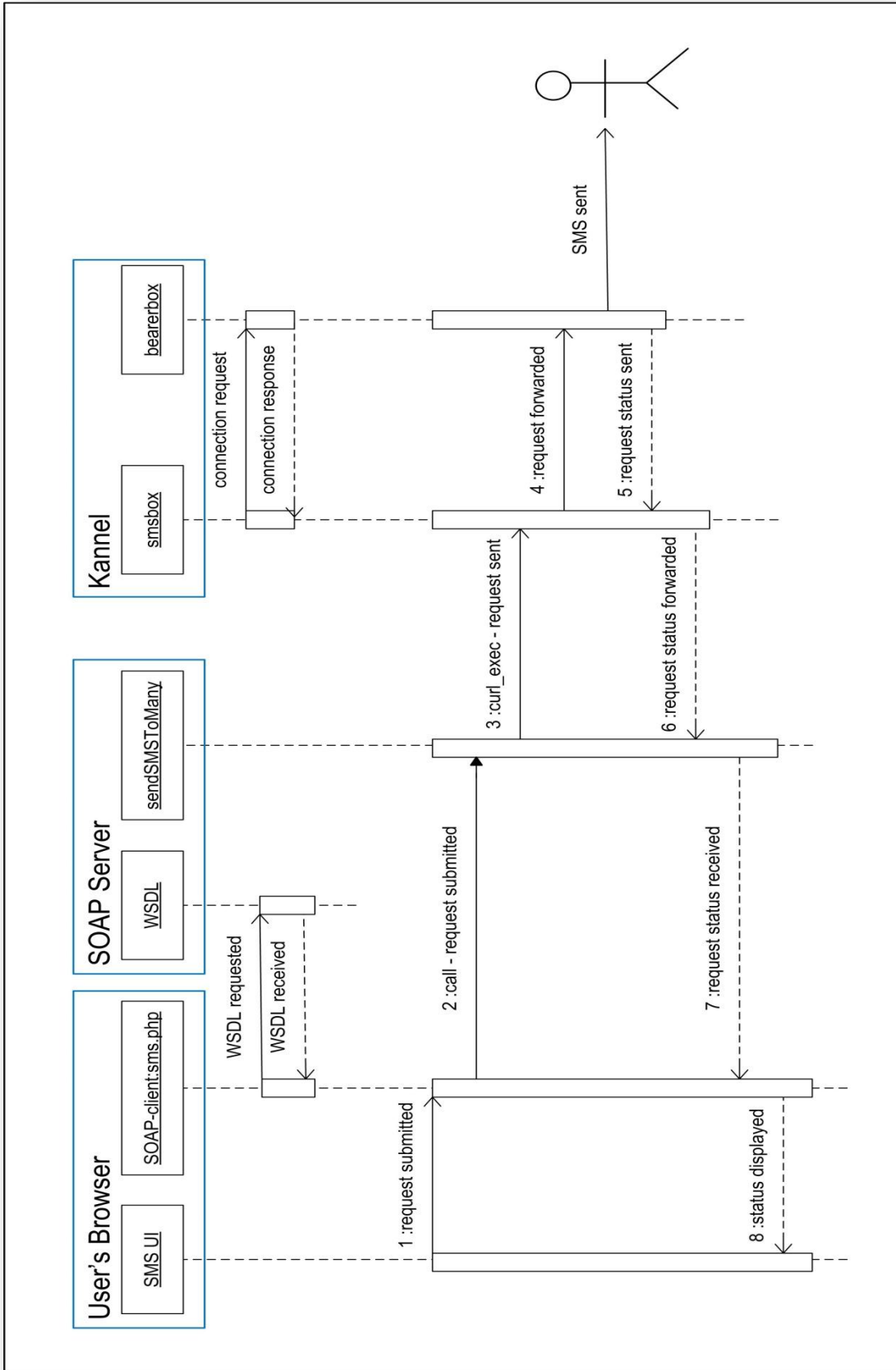


Figure 4.4: SMS Web Service sequence diagram

4.8. MMS Web Service Architecture

MMSs are more complex than SMSs. They involve sending multimedia messages which range from simple text messages to more sophisticated slideshows comprising of text, image and audio clips (Bodic, 2005:207). In order to enjoy MMSs the user must have a device that has an MMS client installed in it and he/she must be registered for the MMS service (Bodic, 2005:218 - 220).

In this project, the MMS Web Service application involves the use of two gateways. It uses Mbuni as the MMS gateway and the *wapbox* in Kannel to push notification messages to the recipient, notifying him/her of the MMS sent. When Mbuni is configured as an MMSC, it stores the MMS whilst waiting for the recipient to retrieve it. The notification sent to the recipient contains the Uniform Resource Locator (URL) the recipient needs in order to retrieve the MMS.

This is illustrated in the two diagrams below. Figure 4.5 shows the different components involved in sending a MMS and Figure 4.6 shows a chain of requests and responses involved in sending a MMS.

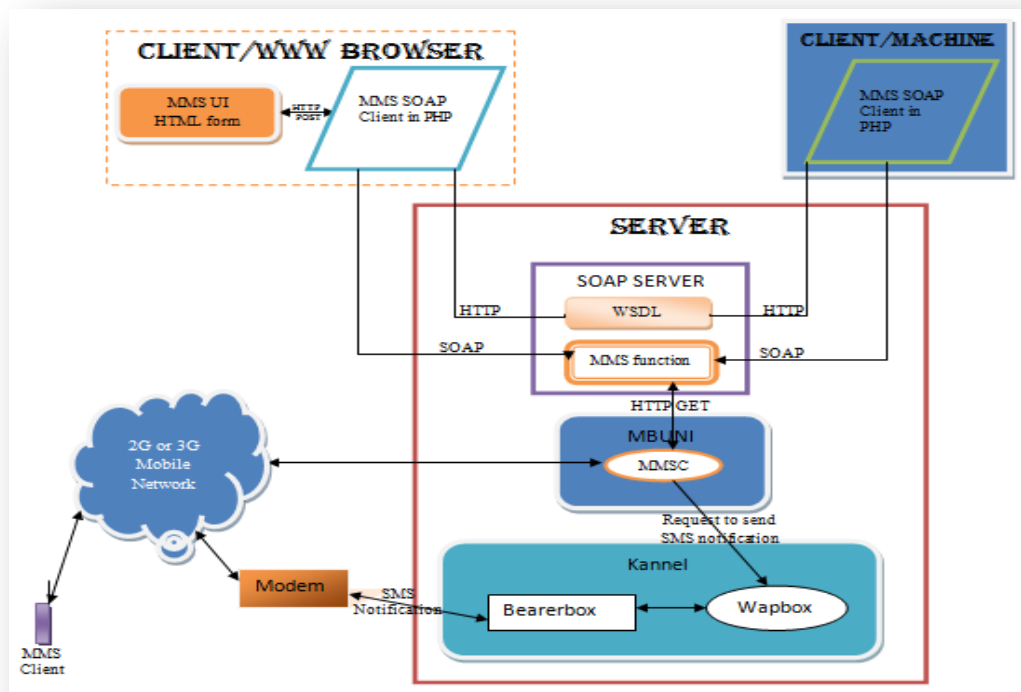


Figure 4.5: MMS Web Service architecture

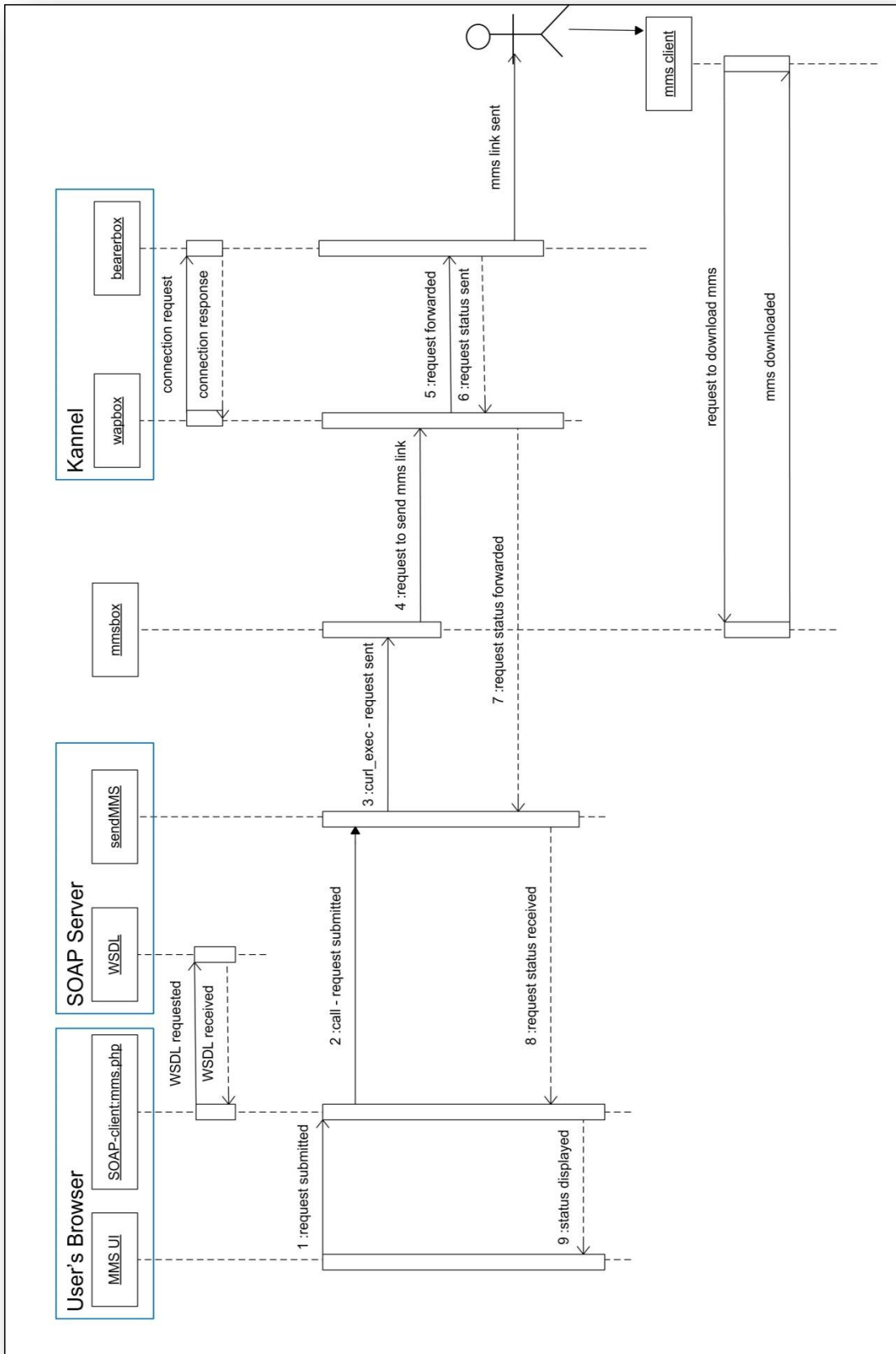


Figure 4.6: MMS Web Service sequence diagram

4.9. Email Web Service Architecture

Bodic (2005:210) states that the Email has now become the universal messaging service for Internet users. Email services have support for group sending, message attachments, automatic message forwarding and many more.

Since Emails are a messaging service for Internet users, this Email Web Service application involves the use of a DNS. Bind9 is the DNS in use and it is discussed under section 3.3.3. Postfix is the SMTP server used for the sending of the emails.

Figure 4.7 shows the different components involved in sending an Email and Figure 4.8 shows a chain of requests and responses involved in sending an Email.

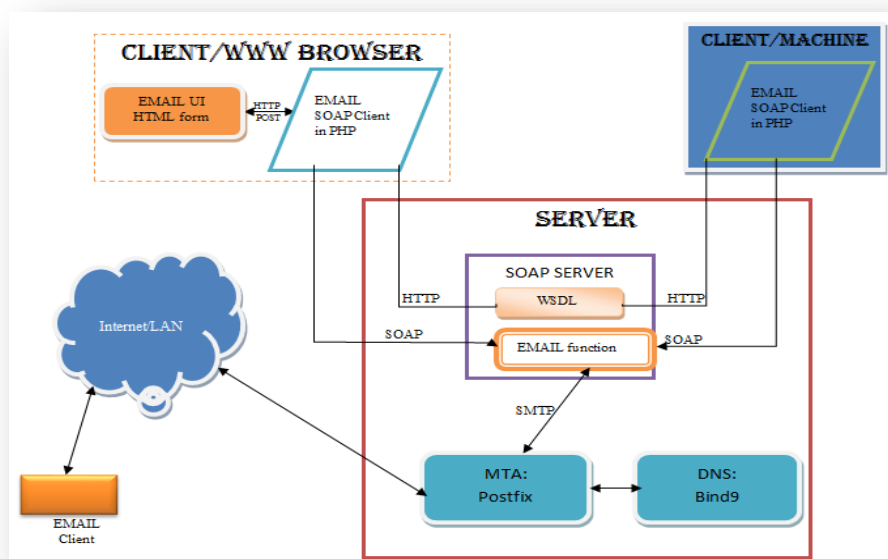


Figure 4.7: Email Web Service Architecture

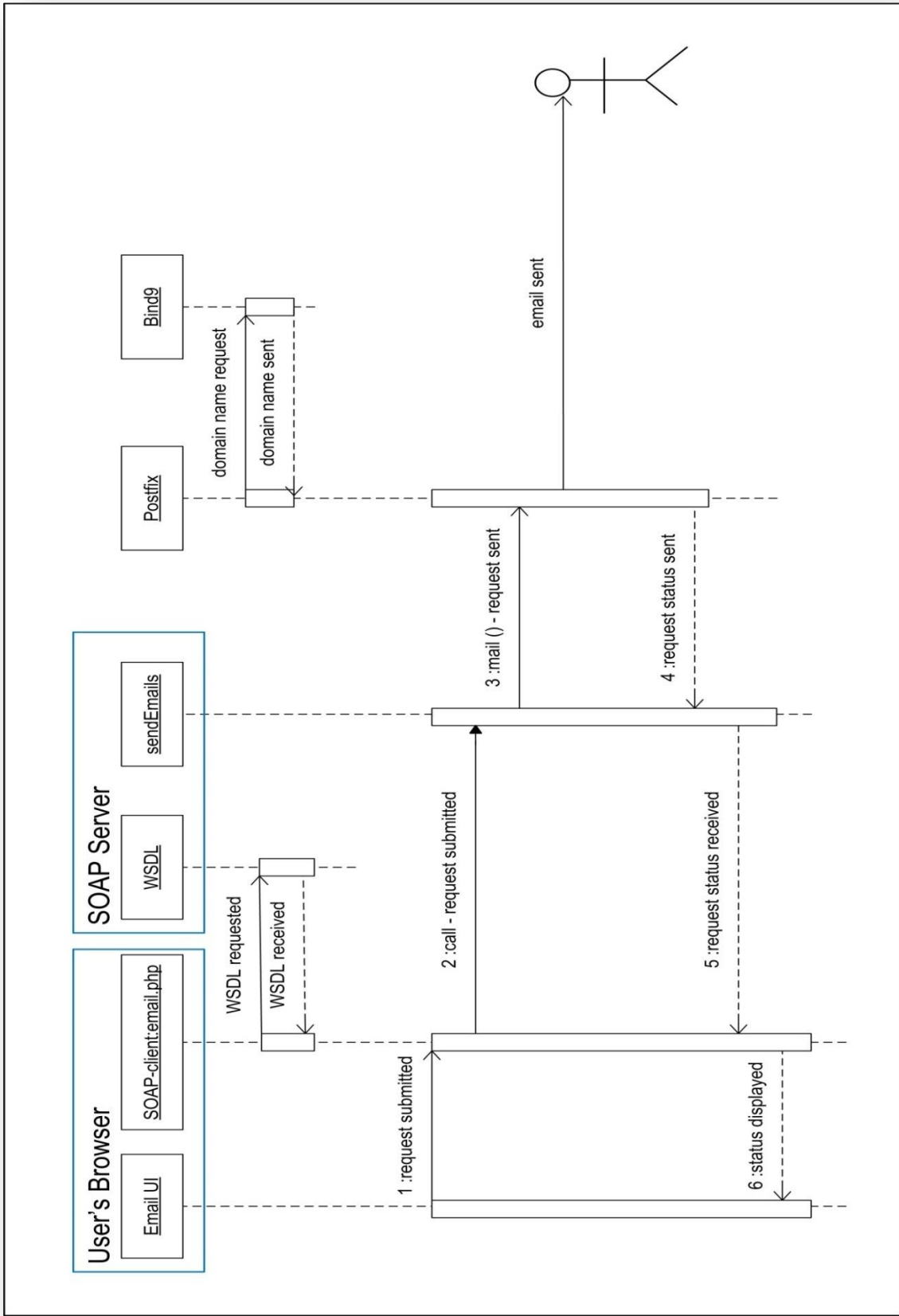


Figure 4.8: Email Web Service sequence diagram

4.10. IM Web Service Architecture

IM is a form of real-time direct text-based communication between two or more parties.

The IM Web Service application allows the system and/or the system administrator to send real-time notification messages to users. This application is not meant to be used by any system users. It is only for system administration purposes. This application is designed to send alerts to administrators and/or users online on the system's XMPP server or on Google Talk (GTalk).

This is illustrated in the two diagrams below. Figure 4.9 shows the different components involved in sending an IM and Figure 4.10 shows a chain of requests and responses involved in sending an IM.

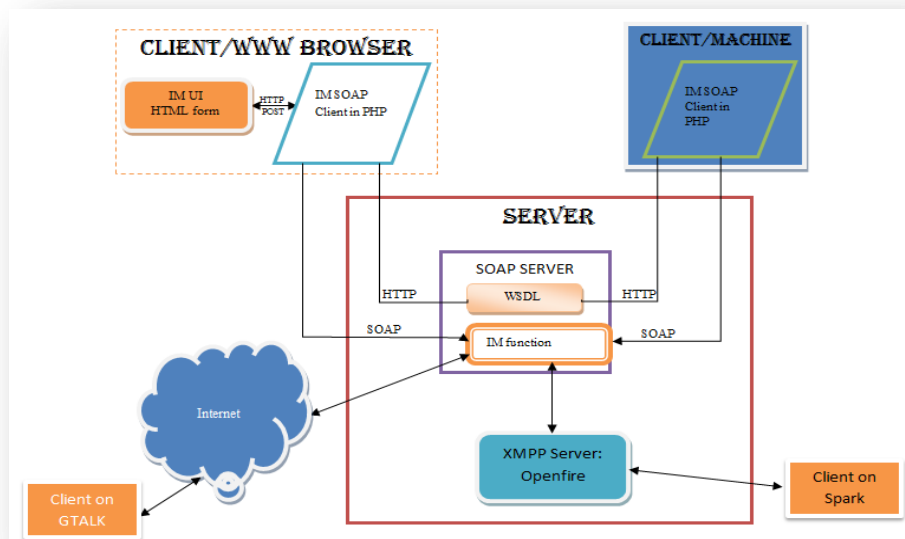


Figure 4.9: IM Web Service Architecture

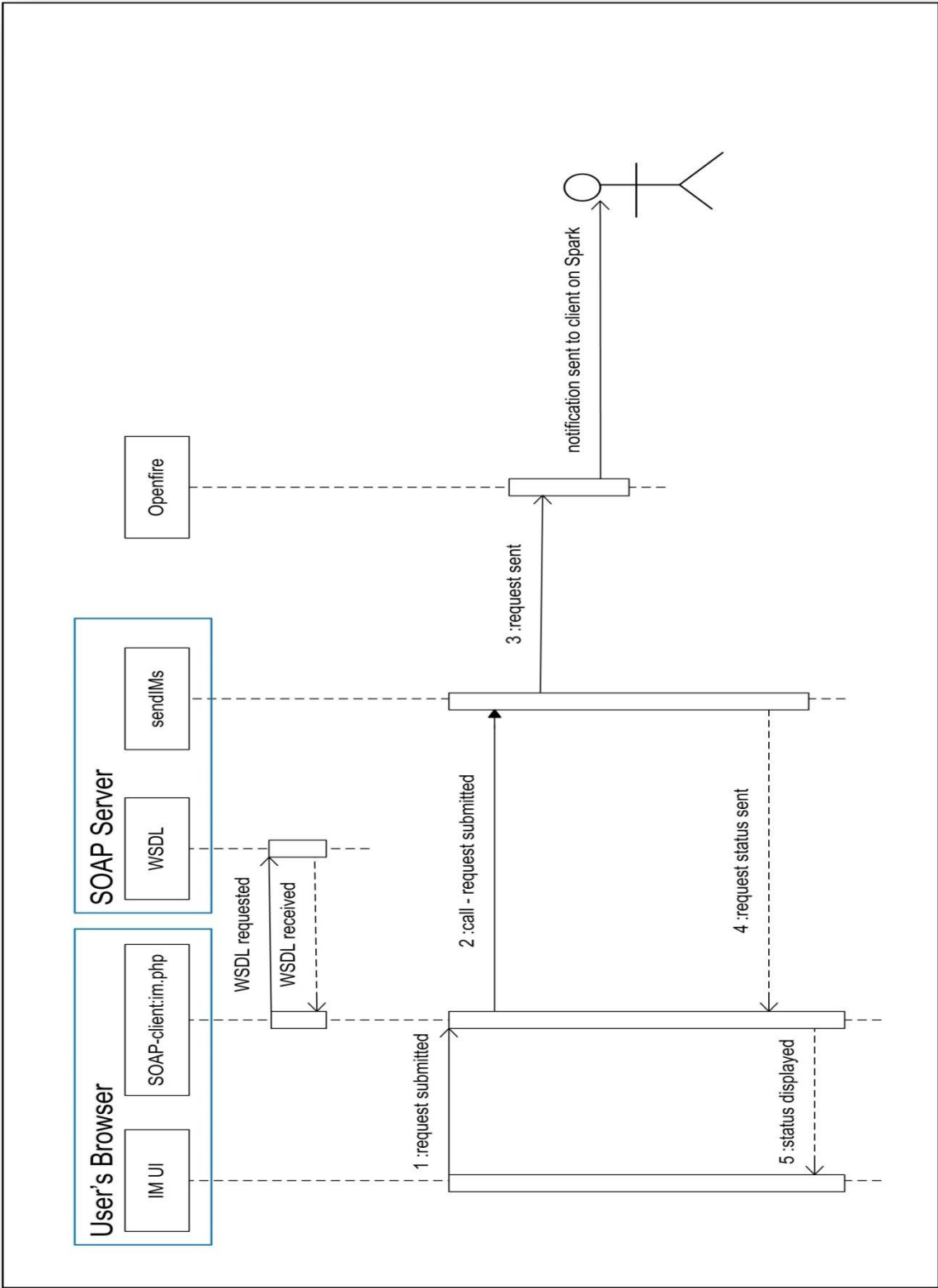


Figure 4.10: IM Web Service sequence diagram

4.11. System's Users

This system has two types of consumers: a person and a machine. In addition, it has two types of users under the category of person: the administrator and the client (referred to as user). The administrator is responsible for managing users' accounts, and he/she is the only person who can make use of the IM UI to send notification messages to users' IM accounts. The administrator can also consume Web Services. Users can register, change passwords, request new passwords (in case the old ones have been forgotten), login, view their credits and consume Web Services.

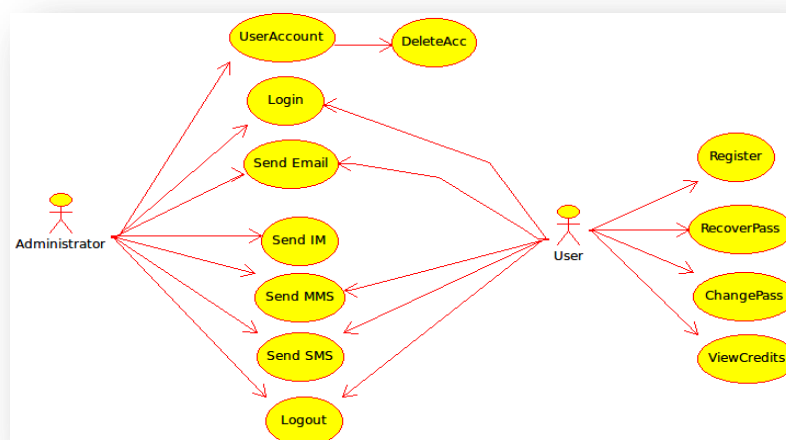


Figure 4.11: Use Case diagram

4.12. Conclusion

This chapter defines the conceptual architecture of this system. It also discusses each component and its architectures. The next chapter will discuss, in detail, how each component is implemented to form the complete system.

CHAPTER

V

5. THE IMPLEMENTATION

5.1. Introduction

The previous chapter discusses the design and representation of the system. This chapter focuses its discussion on the manner in which the different components of the system are implemented. It discusses the SOAP-server, Web Services, SOAP-clients and the UI. In some sections of this chapter, sample codes are shown when necessary. This chapter also discusses the basic configurations and implementation of the SMS and MMS gateway, the DNS, the MTA and the XMPP server.

5.2. SOAP-server

This section discusses the implementation of the SOAP-server.

```
1 <?php
2 require_once("nusoap.php");
3
4 require_once("functions.php");
5
6 $server = new soap_server();
7
8 $server->configureWSDL('Connecting Dwesa People','urn:communicationservices');
9
10 $server->register(
11     'sendSMSToMany',
12     array('to' => 'xsd:string','from' => 'xsd:string', 'message' => 'xsd:string'),
13     array('status' => 'xsd:string'),
14     'urn:communicationservices',
15     'urn:communicationservices#sendSMSToMany',
16     'rpc',
17     'encoded',
18     'Sends the same SMS to multiple phone numbers.'
19 );
20
21 function sendSMSToMany($from, $to, $msg){
22     return new soapval('return','xsd:string',sendSMS($from, $to, $msg));
23 }
24
25 $HTTP_RAW_POST_DATA = isset($HTTP_RAW_POST_DATA) ? $HTTP_RAW_POST_DATA : '';
26 $server->service($HTTP_RAW_POST_DATA);
27 ?>
```

Listing 5.1: Creating a SOAP-server using NuSOAP

The image above shows the implementation of the system's SOAP-server housing only the SMS Web Service. The complete SOAP-server can be viewed in Appendix D.

The bullet points below describe the steps taken to implement the above SOAP-server.

- Line 2: The NuSOAP class, *nusoap.php*, is required first.
- Line 7: A *soap_server* instance, named *\$server*, is created.
- Line 8: Initialization of the WSDL support.
- Lines 10 – 19: Registration of the method to expose. In this case, the SMS Web Service is the method registered.
 - Line 11: The method's name.
 - Line 12: The type and number of input parameters expected from the SOAP-client. This method has four input parameters and they are all strings. The parameters under function call in the SOAP-client must correspond with these otherwise the function call will return an error message.
 - Line 13: The type and number of output parameters. In most instances, there is one output parameter, which can either be of type Boolean, string, etc. In this case, the method returns a string.
 - Lines 14 -15: Respectively, the namespace and SOAPaction.
 - Lines 16 – 17: They state that the method used to call a Web Service is a Remote Procedure Call (rpc) which is encoded (lyingonthecovers.net, 2006).
 - Line 18: The documentation. This line gives a slight description of the Web Service.
- Lines 21 – 23: The method is now defined as a PHP function which calls another function within it; in this case the function called within the method is *sendSMS*. This function is located in a separate file known as *functions.php*, hence line 4 is needed.
- Lines 25 – 26: The request sent to the server is strictly raw data and it is retrieved using `HTTP_RAW_POST_DATA` (Richards, 2006:617).

When the SOAP-server is completely developed using NuSOAP, a user can view and learn about the Web Services in a browser as shown in Figure 5.1 below. The NuSOAP library offers an interface that describes all the functions made available through the Web Services (Nichol, 2004).

To view the NuSOAP interface, the user must type the following URL in his/her browser: *http://localhost/webservices/soap_server.php*. Where:

- Localhost: it is running on localhost.
- Webservices: the directory in which the SOAP-server is located.
- Soap_server.php is the name of the SOAP-server.

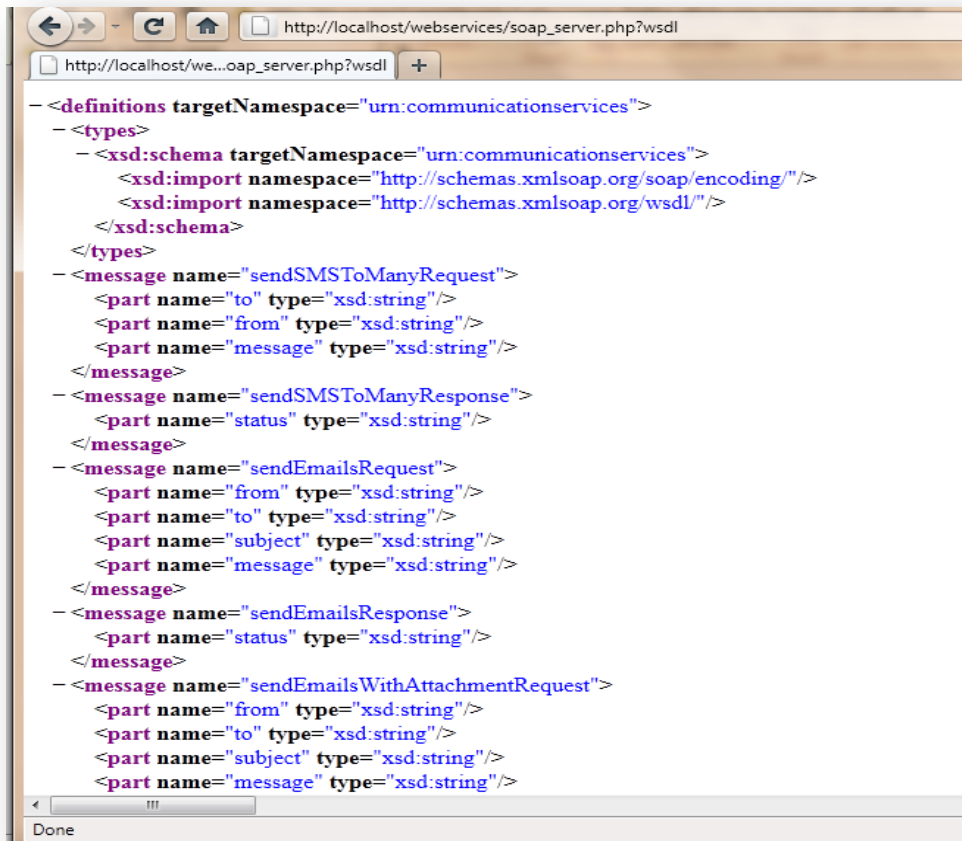


Figure 5.1: NuSOAP interface

This NuSOAP interface shows that there are four methods registered in the SOAP-server. The user can click on each one of them in order to get their descriptions. A screenshot showing the *sendEmails* descriptions can be viewed in Appendix B.

The WSDL document can be viewed in two ways; either by clicking the link provided on the NuSOAP interface or by typing the following URL in the browser:

http://localhost/webservices/soap_server.php?wsdl.



```
-<definitions targetNamespace="urn:communicationservices">
  -<types>
    -<xsd:schema targetNamespace="urn:communicationservices">
      <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
      <xsd:import namespace="http://schemas.xmlsoap.org/wsdl/" />
    </xsd:schema>
  </types>
  -<message name="sendSMSToManyRequest">
    <part name="to" type="xsd:string"/>
    <part name="from" type="xsd:string"/>
    <part name="message" type="xsd:string"/>
  </message>
  -<message name="sendSMSToManyResponse">
    <part name="status" type="xsd:string"/>
  </message>
  -<message name="sendEmailsRequest">
    <part name="from" type="xsd:string"/>
    <part name="to" type="xsd:string"/>
    <part name="subject" type="xsd:string"/>
    <part name="message" type="xsd:string"/>
  </message>
  -<message name="sendEmailsResponse">
    <part name="status" type="xsd:string"/>
  </message>
  -<message name="sendEmailsWithAttachmentRequest">
    <part name="from" type="xsd:string"/>
    <part name="to" type="xsd:string"/>
    <part name="subject" type="xsd:string"/>
    <part name="message" type="xsd:string"/>
  </message>
</definitions>
```

Listing 5.2: Part of the WSDL document

The SOAP-server is of no use if it does not contain any Web Service.

The sections which follow explore the four Web Services housed in the SOAP-server. The registration of each method can be viewed in Appendix D.

5.3. SMS Web Service

This section explains, in detail, how the SMS Web Service is implemented. It discusses, step-by-step, how each component involved in the SMS Web Service application is installed and configured to meet the research objectives.

5.3.1. The Method

The SMS Web Service is registered in the SOAP-server as *sendSMStoMany* and it uses a PHP function called *sendSMS* to connect the system to Kannel. *sendSMS* is the function responsible for sending out SMSs and returning a response to the client via *sendSMStoMany*.

5.3.2. The Gateway: Kannel

Installing Kannel can be done in two different ways on an Ubuntu machine.

Using the Synaptic package manager to download and install it automatically.

Downloading Kannel from <http://www.kannel.org/download.shtml> and manually installing it.

There is a manual on Kannel's website that gives a tutorial on how to install and configure Kannel either as a wap Gateway or a SMS gateway.

Upon successful installation, the following Kannel's programs are installed in `/usr/local/sbin`:

- Bearerbox
- Smsbox
- Wapbox

Kannel's package comes with a default configuration file located in the `/doc/` directory. The default configuration file is often named *kannel.conf* and requires only minor changes. This file can be viewed in Appendix C.

When configuring Kannel as a SMS gateway, there are five groups that must be present in *kannel.conf*. The groups are the core group, the *smc* group, the *smsbox* group, the *sendsms-user* group and the *sms-service* group (Wandschneider, 2007).

There is another important configuration file that is needed in order for Kannel to run properly. The *modems.conf* is a default configuration file that defines all types of modems via which Kannel can connect to a SMSC. This file is simply included at the bottom of *kannel.conf*. A copy of *modems.conf* can be viewed in Appendix C.

It is good practice to move the configuration file to a proper directory such as `/etc/kannel/`. Since Kannel's programs are stored in `/usr/local/sbin`, the following is done to test the configuration file:

```
cd /usr/local/sbin/  
bearerbox /etc/kannel/kannel.conf  
smsbox /etc/kannel/kannel.conf.
```

The *bearerbox* must be started before any other application within Kannel can be started. If the *bearerbox* runs correctly to the point of connection to a SMSC, this means that the basic configurations are properly done. In the case of errors, the execution is terminated and the error(s) are displayed on the screen.

The *smsbox* is only started after the *bearerbox* is successfully running. The *smsbox* establishes connection with the *bearerbox* and waits for HTTP requests from users.

These two are often started manually and can be set to start automatically. To accomplish that, cron jobs have been created to start the *bearerbox* and the *smsbox* at boot time.

```
GNU nano 2.2.2                               File: /tmp/crontab.K2Ax5w/crontab
# m h dom mon dow  command
@reboot      /etc/sbin/bearerbox /etc/kannel/kannel.conf && /etc/sbin/smsbox /etc/kannel/kannel.conf
@reboot      /usr/local/bin/mmsbox /etc/mbuni/mmsbox.conf
```

Figure 5.2: *bearerbox* and *smsbox* cron jobs

5.3.3. Connecting to Kannel

Before connecting to Kannel's interface, the SMS push must be enabled. This is done by setting up a *sendsms-port* in the *smsbox* group and by properly configuring the *sendsms-user* group.

```
41 # SMSBOX SETUP
42
43 group = smsbox
44 bearerbox-host = localhost
45 smsbox-id = box
46 sendsms-port = 13013
```

Listing 5.3: *smsbox* group.

```
56 # SEND-SMS USERS
57
58 group = sendsms-user
59 username = jimmy
60 password = junior
61 user-allow-ip = "*.*.*.*"
62 max-messages = 3
63 concatenation = true
```

Listing 5.4: *sendsms-user* group

There can be more than one *sendsms-user* group. This group defines an account which can be used to push SMSs via an HTTP interface.

The URL used reads as follows:

```
http://host:port/cgi-
bin/sendsms?username=$user&password=$pass&from=$from&to=$p&text=$msg
```

Where:

- Host: In many instances it is set to localhost
- Port: *sendsms-port* registered under the *smsbox* group
- \$user: username registered under *sendsms-user* group
- \$pass: password registered under *sendsms-user* group
- \$from: the number of the sender
- \$to: the number of the recipient
- \$text: is the message to send

A successful implementation of the SMS Web Service application is of great importance because the MMS Web Service application, discussed in the next section, depends on it.

5.4. MMS Web Service

This section explains, in detail, how the MMS Web Service is implemented. It discusses, step-by-step, how each component involved in the MMS Web Service application is installed and configured to meet the research objectives.

5.4.1. The method

The MMS Web Service is registered in the SOAP-server as *sendMMSs* and it uses a PHP function called *sendMMS* to connect the system to Mbuni. *sendMMS* is the function responsible for sending out MMSs and returning a response to the client via *sendMMSs*.

5.4.2. The gateway: Mbuni

Mbuni is not found in the synaptic package manager. So it is manually downloaded from <http://www.mbuni.org/downloads.shtml>.

Mbuni comes with a user guide manual that explains the steps to follow in order to install Mbuni.

Mbuni can be run as a MMSC or a VAS Gateway and, for this reason, it has four programs, which are installed in */usr/local/bin* and two configuration files, by default. The four programs are *mmsrealy*, *mmsproxy*, *mmsfromemail* and *mmsbox*.

The first two programs are used to run Mbuni as a MMSC and the third one is needed to convert an MMS from an email sender. The fourth program is used to run Mbuni as a VAS gateway. In the case of this project, Mbuni is used as a VAS gateway and only one configuration file is used.

Mbuni has two configuration files which are found in the */doc/examples/* directory. The first file is named *mmsc.conf* and the second one is named *mmsbox.conf*; the latter can be viewed in Appendix C. Since Mbuni is used as a VAS Gateway in this project, the second configuration file is required.

The *mmsbox.conf* has five groups in it. These are: the *core* group, the *mbuni* group, the *mmsc* group, the *mms-service* group and the *send-mms-user* group.

5.4.3. Connecting to Mbuni

In order to send MMSs via an HTTP interface, the send MMS service must be enabled. This is done by setting up a *sendmms-port* in the *mbuni* group and by properly configuring the *send-mms-user* group.

```
6 group = mbuni
7 storage-directory = /var/spool/mbuni
8 max-send-threads = 5
9 maximum-send-attempts = 50
10 default-message-expiry = 360000
11 queue-run-interval = 5
12 send-attempt-back-off = 300
13 sendmms-port = 10001
```

Listing 5.5: *mbuni* group.

```
42 group = send-mms-user
43 username = jimmy
44 password = junior
45 faked-sender = 100
```

Listing 5.6: *send-mms-user* group

The following is the URL used to send MMSs using an HTTP interface.

```
http://host:port/?username=$username&password=$pass&to=$to/TYPE=PLMN&subject=$subject&text=$mms
```

Where:

- Host: localhost
- Port: *sendmms-port* registered under the Mbuni group.
- \$username: username registered under *send-mms-user* group.
- \$password: password registered under *send-mms-user* group.

Since Mbuni is configured as a VAS gateway, the *sendmms* port must be configured in order to make use of the send MMS service. This service is called up the same way as the send SMS service but, this time, requests are sent to the *sendmms* port on the VAS Gateway interface.

A *wapbox* configuration has to be added to *kannel.conf* in order for Mbuni to send notification messages to the MMSs recipient of the MMS.

```
23 group = wapbox
24 bearerbox-host = localhost
25 log-file = "/tmp/wapbox.log"
26 syslog-level = none
27 access-log = "/tmp/wapaccess.log"
28 timer-freq = 10
29 map-url = "http://mmsc/* http://localhost:1981/*"
```

Listing 5.7: *wapbox* group

5.5. Email Web Service

This section explains, in detail, how the Email Web Service is implemented. It discusses, step-by-step, how each component involved in the Email Web Service application is installed and configured to meet the research objectives.

5.5.1. The method

The Email Web Service is registered in the SOAP-server as *sendEmails* and it uses a PHP function called *sendEmail* to connect the system to Postfix. This function is also responsible for sending out Emails and returning a response to the client via *sendEmails*.

5.5.2. The DNS

Bind9 has been configured to run as a local DNS. There are five files that have been configured in order for the DNS to successfully resolve names. The configuration file shown below is the *db.dwesaproject.com* zone file. The other files can be viewed in Appendix C.

```

1 ;
2 ; BIND data file for local loopback interface
3 ;
4 $TTL      604800
5 @      IN  SOA ns.dwesaproject.com. root.dwesaproject.com. (
6          |          |          |          |          |          |
7          |          |          |          |          |          |
8          |          |          |          |          |          |
9          |          |          |          |          |          |
10         |          |          |          |          |          |
11         |          |          |          |          |          |
12         |          |          |          |          |          |
13         |          |          |          |          |          |
14         |          |          |          |          |          |
15         |          |          |          |          |          |
16         |          |          |          |          |          |
17         |          |          |          |          |          |
18         |          |          |          |          |          |
19         |          |          |          |          |          |
20         |          |          |          |          |          |
21         |          |          |          |          |          |
22         |          |          |          |          |          |
23         |          |          |          |          |          |
24         |          |          |          |          |          |
25         |          |          |          |          |          |
26         |          |          |          |          |          |
27         |          |          |          |          |          |
28         |          |          |          |          |          |
29         |          |          |          |          |          |
30         |          |          |          |          |          |
31         |          |          |          |          |          |
32         |          |          |          |          |          |
33         |          |          |          |          |          |
34         |          |          |          |          |          |
35         |          |          |          |          |          |
36         |          |          |          |          |          |
37         |          |          |          |          |          |
38         |          |          |          |          |          |
39         |          |          |          |          |          |
40         |          |          |          |          |          |
41         |          |          |          |          |          |
42         |          |          |          |          |          |
43         |          |          |          |          |          |
44         |          |          |          |          |          |
45         |          |          |          |          |          |
46         |          |          |          |          |          |
47         |          |          |          |          |          |
48         |          |          |          |          |          |
49         |          |          |          |          |          |
50         |          |          |          |          |          |
51         |          |          |          |          |          |
52         |          |          |          |          |          |
53         |          |          |          |          |          |
54         |          |          |          |          |          |
55         |          |          |          |          |          |
56         |          |          |          |          |          |
57         |          |          |          |          |          |
58         |          |          |          |          |          |
59         |          |          |          |          |          |
60         |          |          |          |          |          |
61         |          |          |          |          |          |
62         |          |          |          |          |          |
63         |          |          |          |          |          |
64         |          |          |          |          |          |
65         |          |          |          |          |          |
66         |          |          |          |          |          |
67         |          |          |          |          |          |
68         |          |          |          |          |          |
69         |          |          |          |          |          |
70         |          |          |          |          |          |
71         |          |          |          |          |          |
72         |          |          |          |          |          |
73         |          |          |          |          |          |
74         |          |          |          |          |          |
75         |          |          |          |          |          |
76         |          |          |          |          |          |
77         |          |          |          |          |          |
78         |          |          |          |          |          |
79         |          |          |          |          |          |
80         |          |          |          |          |          |
81         |          |          |          |          |          |
82         |          |          |          |          |          |
83         |          |          |          |          |          |
84         |          |          |          |          |          |
85         |          |          |          |          |          |
86         |          |          |          |          |          |
87         |          |          |          |          |          |
88         |          |          |          |          |          |
89         |          |          |          |          |          |
90         |          |          |          |          |          |
91         |          |          |          |          |          |
92         |          |          |          |          |          |
93         |          |          |          |          |          |
94         |          |          |          |          |          |
95         |          |          |          |          |          |
96         |          |          |          |          |          |
97         |          |          |          |          |          |
98         |          |          |          |          |          |
99         |          |          |          |          |          |
100        |          |          |          |          |          |

```

Listing 5.8: *db.dwesaproject.com* zone file

Now that the DNS is running properly, the next step is to configure a MTA: Postfix

5.5.3. The MTA: Postfix

The Ubuntu server edition has Postfix installed automatically when the OS is installed. The only thing that needs to be done is to configure Postfix to fit the system's requirements.

There are many ways in which Postfix can be configured depending on the environment and/or the problem to solve (Postfix, n.d.). For this system, Postfix is configured to run on a local network.

The Postfix configuration file is named *main.cf*. It is situated in */etc/postfix/* and it is configured as follows:

```

1 myhostname = mail.dwesaproject.com
2 mydestination = $myhostname localhost.$mydomain localhost $mydomain
3 mynetworks = 127.0.0.0/8 172.20.56.0/24
4 inet_interfaces = all
5 relayhost = $mydomain
6 home_mailbox = Maildir/

```

Listing 5.9: *main.cf*

Now that Postfix is configured to send and receive emails, its configuration file must be tested to verify whether there are no errors. This is done by restating Postfix from the command line, and if there are errors, it either fails to restart or it restarts and is ready to send and receive emails.

Since this system involves remote users, there is a need to install a Post Office Protocol (POP) or Internet Message Access Protocol (IMAP) server to allow those users to send emails. The application chosen for this task is Dovecot.

5.5.4. POP/IMAP Server: Dovecot

Dovecot is installed from the synaptic package manager and only one line in its configuration file, named *dovecot.conf*, is changed as follows:

```
#Open dovecot.conf using nano
nano /etc/dovecot.conf
#uncomment the following line
protocols = imap imaps pop3 pop3s
```

5.6. IM Web Service

This section explains, in detail, how the IM Web Service is implemented. It discusses, step-by-step, how each component involved in the IM Web Service application is installed and configured to meet the research objectives.

5.6.1. The Method

The SMS Web Service is registered in the SOAP-server as *sendIMs* and it uses a PHP function called *sendIM* and the XMPPHP library to connect the system to Openfire. This function is also responsible for sending out IM notifications and returning a response to the client via *sendIMs*.

5.6.2. The XMPP Server

Openfire can be downloaded from the following URL:

http://www.igniterealtime.org/downloads/download-landing.jsp?file=openfire/openfire_3_6_4.tar.gz

Once Openfire has been successfully installed, a few configurations need to take place to meet the system's requirements.

5.6.3. Configuring Openfire and Apache HTTP Server

Minor configurations are done in Openfire and Apache HTTP server in order to meet the system's requirements.

- Since the IM notifications are intended only for online users, on the admin screen of Openfire, in Server settings, the Offline Message policy must be set to Drop. This disallows the XMPP server to keep messages sent to users when they are offline (Werdmuller, 2010).
- In order to prevent unauthorized connections from causing confusion, the Service Enabled must be set to Disabled and Allowed to Connect to the White List. This is done in Server settings, under Server-to-Server, (Werdmuller, 2010).
- Werdmuller (2010) states that Openfire maintains an HTTP binding URL for access over BOSH at <http://localhost:7070/http-bind>. To use this on port 80, the proxy module in Apache's configuration file must be enabled. This allows Apache to forward a URL to this location. The following must be done to enable the proxy module in Apache:
 - In the Apache configuration file the following lines, `LoadModule proxy_http_module modules/mod_proxy_http.so` and `LoadModule proxy_module modules/mod_proxy.so`, have to be located and uncommented:

```
116 LoadModule proxy_module modules/mod_proxy.so
117 #LoadModule proxy_ajp_module modules/mod_proxy_ajp.so
118 #LoadModule proxy_balancer_module modules/mod_proxy_balancer.so
119 #LoadModule proxy_connect_module modules/mod_proxy_connect.so
120 #LoadModule proxy_ftp_module modules/mod_proxy_ftp.so
121 LoadModule proxy_http_module modules/mod_proxy_http.so
```

Listing 5.10: Lines 116 and 121 have been uncommented in `httpd.conf`

- After uncommenting the two lines above, the following lines must be added at the bottom of the Apache configuration file:

```
509 # XMPP proxy rule
510 ProxyRequests Off
511 ProxyPass /xmpp-httpbind http://127.0.0.1:7070/http-bind/
512 ProxyPassReverse /xmpp-httpbind http://127.0.0.1:7070/http-bind/
```

Listing 5.11: Setting Apache XMPP proxy rule

5.6.4. The IM Client

Now that Openfire and Apache HTTP Server are ready for the task, there is a need for an IM client application so that users can be online and receive the notifications.

There are a number of IM client applications that can be used XMPP client but Spark is the one that is used in this project.

Sparks screenshots can be viewed in Appendix B.

5.6.5. Connecting to the XMPP server

The SOAP-server connects to the XMPP server through a server-side application created using XMPPHP library. In the case of sending IM notifications to a recipient's GTalk account, a server application called *gtalk.php* is used. This application is simply a PHP file that contains a class called *GTalk*. *GTalk* contains the following PHP functions:

- *__construct()*: this constructor defines the parameters needed to create a new connection to the XMPP server (talk.google.com).
- *connect()*: this function is responsible for establishing a connection to the XMPP server.
- *disconnect()*: this function is used to disconnect from the XMPP server.
- *send_message(\$to, \$msg)*: this function takes two parameters and is responsible for sending the IM notifications.

The same process is used to send IM notifications to a user on Spark except for the fact that the values of the parameters needed in the constructor are different. As a result, there is a second file named *spark.php* which contains a class called *Spark*.

The two classes can be viewed in Appendix D.

Now that the SOAP-server and the Web Services are in place, users have to be able to make use of them. The first objective of this project focuses on machine-to-machine and machine-to-person types of communication, whereas the second objective focuses on a person-to-person type of communication.

In order to satisfy objective one, a SOAP-client must be implemented. Though all the Web Services are housed in one SOAP-server, each Web Service has its own SOAP-client. And in order to satisfy objective two, a user UI laid on top of the SOAP-client must be implemented. The next two sections discuss the implementation of a SOAP-client and a UI.

5.7.SOAP-Client

This section discusses the implementation of the SMS Web Service SOAP-client. This SOAP-client is called *sms.php*. All the other SOAP-clients can be viewed in Appendix D.


```

1 <?php
2 require_once('nusoap.php');
3 $from = $_POST["from"];
4 $to = $_POST["to"];
5 $message = $_POST["smsmessage"];
6
7 $client = new nusoap_client('http://localhost/webservices/soap_server.php?wsdl', true);
8 $err = $client->getError();
9 if ($err) {
10     echo '<h2>Constructor error</h2><pre>' . $err . '</pre>';
11     echo '<h2>Debug</h2><pre>' . htmlspecialchars($client->getDebug(), ENT_QUOTES) . '</pre>';
12     exit();
13 }
14 $result = $client->call(
15     'sendSMSToMany',
16     array('from' => $from, 'to' => $to, 'message' => $message),
17     'urn:communicationservices',
18     'urn:communicationservices#SendSMSToMany'
19 );
20 if ($client->fault) {
21     echo '<h2>Fault (Expect - The request contains an invalid SOAP body)</h2><pre>';
22     print_r($result);
23     echo '</pre>';
24 } else {
25     $err = $client->getError();
26     if ($err) {
27         echo '<h2>Error</h2><pre>' . $err . '</pre>';
28     } else {
29         echo '<h2>Result</h2><pre>'; print_r($result); echo '</pre>';
30     }
31 }
32 ?>

```

Listing 5.12: Creating the SMS SOAP-client using NuSOAP

The bullet points below describe the steps it takes to implement the above SOAP-server.

- Line 2: The NuSOAP class, *nusoap.php*, is required first.
- Line 7: A *nusoap_client* instance, named *\$client*, is created. Since *\$client* uses WSDL, the location of the WSDL document is specified.
- Lines 8 – 12: These lines check whether *\$client* is properly constructed. If not, an error message is displayed and the execution is stopped with error messages.
- Lines 20 – 23: These lines check whether the call was properly made or not. If not the execution is stopped with error messages.

- Lines 25 – 29: These lines display the response from the function call.

Lines 14 – 19 are responsible for calling the Web Service that the SOAP-client is designed for. Line 15 displays the name of the Web Service which, in this case, is *sendSMStoMany*. This Web Service is defined as a PHP function that has three parameters. This is why the SOAP-client sends out three parameters during the call. The names of these parameters must be identically defined in the SOAP-client as they are defined in the SOAP-server; failing which, the call returns an error message.

These parameters get their values in two different ways. A SOAP-client is involved in both ways. This project has two objectives to meet. In the case of objective one, a UI is not needed. So the values of the parameters are pre-set. For instance; in the case of the network monitoring system that sends out the same message to the system administrator, who has a permanent contact number, each time the network is down, the SOAP-client's call looks like the following:

```
40     $result = $client->call(
41         'sendSMStoMany',
42         array('from' => $from, 'to' => $to, 'message' => $message),
43         'urn:communicationservices',
44         'urn:communicationservices#SendSMStoMany'
45     );
```

Listing 5.13: SMS SOAP-Client's call function.

The *from* parameter can be left empty in this case. When Kannel receives the request to send out the SMS and realizes that the *from* parameter is empty, it automatically inserts the default number recorded in *kannel.conf* under the *smsbox* group. The default number is the number of the SIM card in use in the modem.

The second objective of this research requires a UI. The second objective caters for people who have little or no knowledge of Web Services and how they are consumed. The UI communicates with the SOAP-server and consumes the Web Service via a SOAP-client.

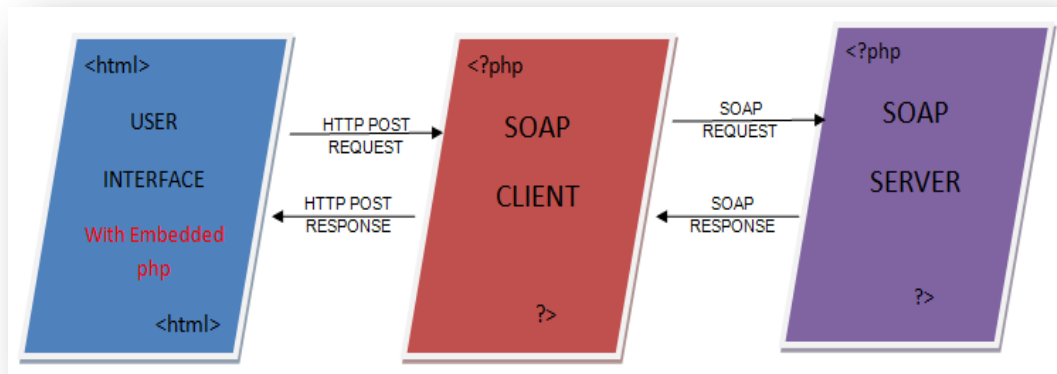


Figure 5.3: Interaction between the UI and SOAP-server via a SOAP-client.

The next section discusses the implementation of Dwesa’s one-stop-shop UI components in its entirety.

5.8. The login interface

When the user types the following URL: <http://localhost/webservices/>² into the browser, one would expect to navigate straight to the welcome page but he/she will be redirected to <http://localhost/webservices/login.php>. This is due to the fact that the welcome page (discussed under section 5.10) offers direct links to the Web Services, and only registered users can access it. This happens because of a file named *auth.php* which is included at the top of the welcome page, in the coding. This file ensures that only registered users can access the system’s web pages upon successful login.s

²Since the welcome page is named “index.php”, typing <http://localhost/webservices/> or <http://localhost/webservices/index.php> gives the same result.

```

1 <?php
2     require_once('includes/funct.php');
3     if (!isset($_SESSION['logged'])){
4         redirect_to("login.php");
5     }
6     ?>
7

```

Listing 5.14: The *auth.php* code

```

<?php
    include ('includes/header.php');
    include ('includes/auth.php');
?>
<div class="welcome">
<table width="600" border="1" cellspacing="0" cellpadding="0">
<tr>
<td class="icon"><a href="sms.php" target="_self"></a></td>
<td class="icon"><a href="mms.php" target="_self" ></a></td>
<td class="icon"><a href="email.php" target="_self" ></a></td>
</tr>
</table>
</div>
<?php include ('includes/footer.php')?>
</body>
</html>

```

Listing 5.15: *index.php*

The login page, *login.php*, allows users to login into the system by supplying their username and password. In this case, the user can supply either his/her username or cellphone number. The system verifies whether the information supplied is in the database. If the information is found in the database, the system logs the user in. If the information is not found in the database a warning message is returned to the user. Non-registered users can register by clicking on the *register* link supplied on the login page. There are two additional links at the bottom of the login page which are links to UIs where users can change their passwords or recover them in the case of loss.

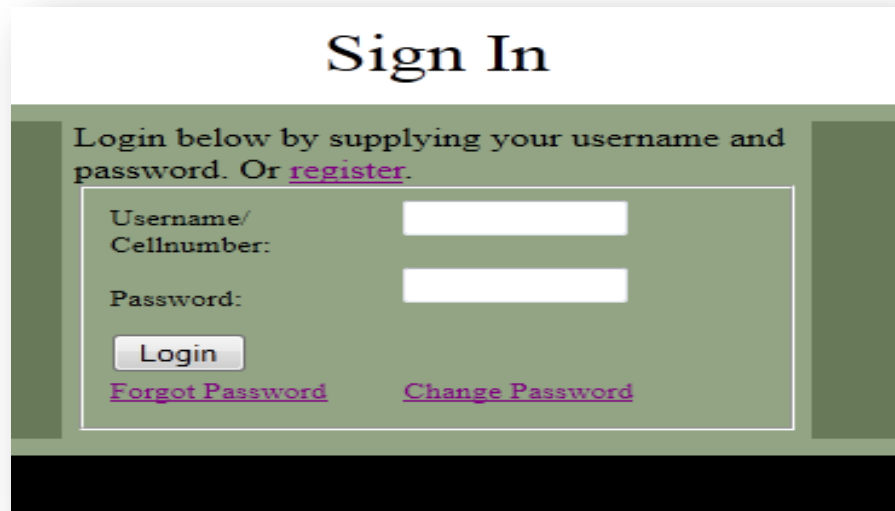


Figure 5.4: Login page UI

- *Forgot password*: this link directs the user to a page that allows him/her to reset the password when he/she has forgotten it. The user is asked to supply the email address he/she submitted upon registration and then submit the form. The system then checks the user's database to verify whether the email address supplied exists in the database. This is one of the reasons why, during registration, the system checks whether the email address supplied is not already in use. Upon successful submission, the system generates a random password for the user, sends the password to the user via email and updates the user's password in the database. When the user receives the new password via email, he/she can then change it to one he/she can easily remember.

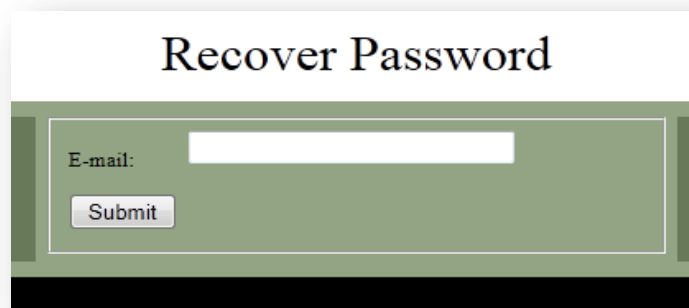


Figure 5.5: Recover password UI

- *Change password*: this link directs the user to a page that allows him/her to change his/her password. The user is asked to supply his/her username and current password, and he/she enters the new password twice. The system then checks the database to verify whether the username supplied actually exists and if the password is correct. Upon successful submission, the system changes the user's password.

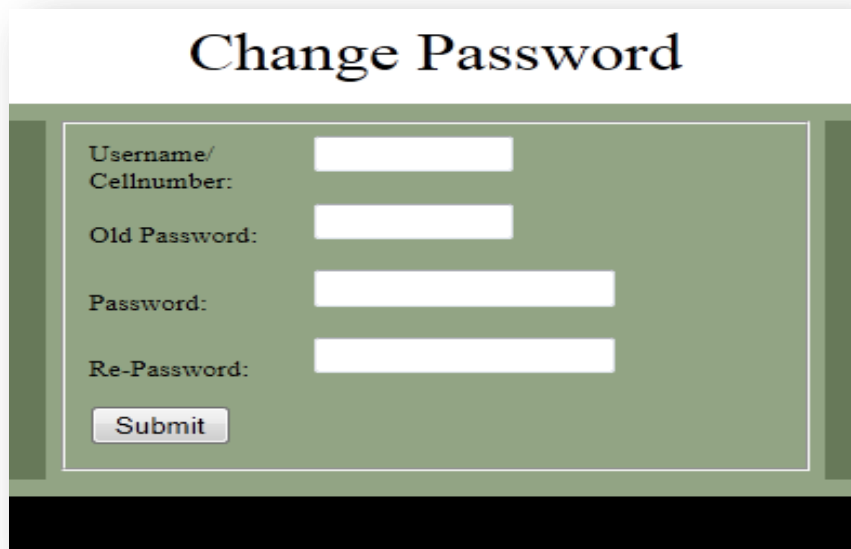
The image shows a web form titled "Change Password" in a large, bold, black serif font at the top center. Below the title is a light green rectangular area containing the form fields. On the left side of this area, the labels "Username/ Cellnumber:", "Old Password:", "Password:", and "Re-Password:" are listed vertically in a black serif font. To the right of each label is a white rectangular input field. At the bottom left of the green area is a grey rectangular button with the word "Submit" in black text. The entire form is set against a white background with a dark green border at the bottom.

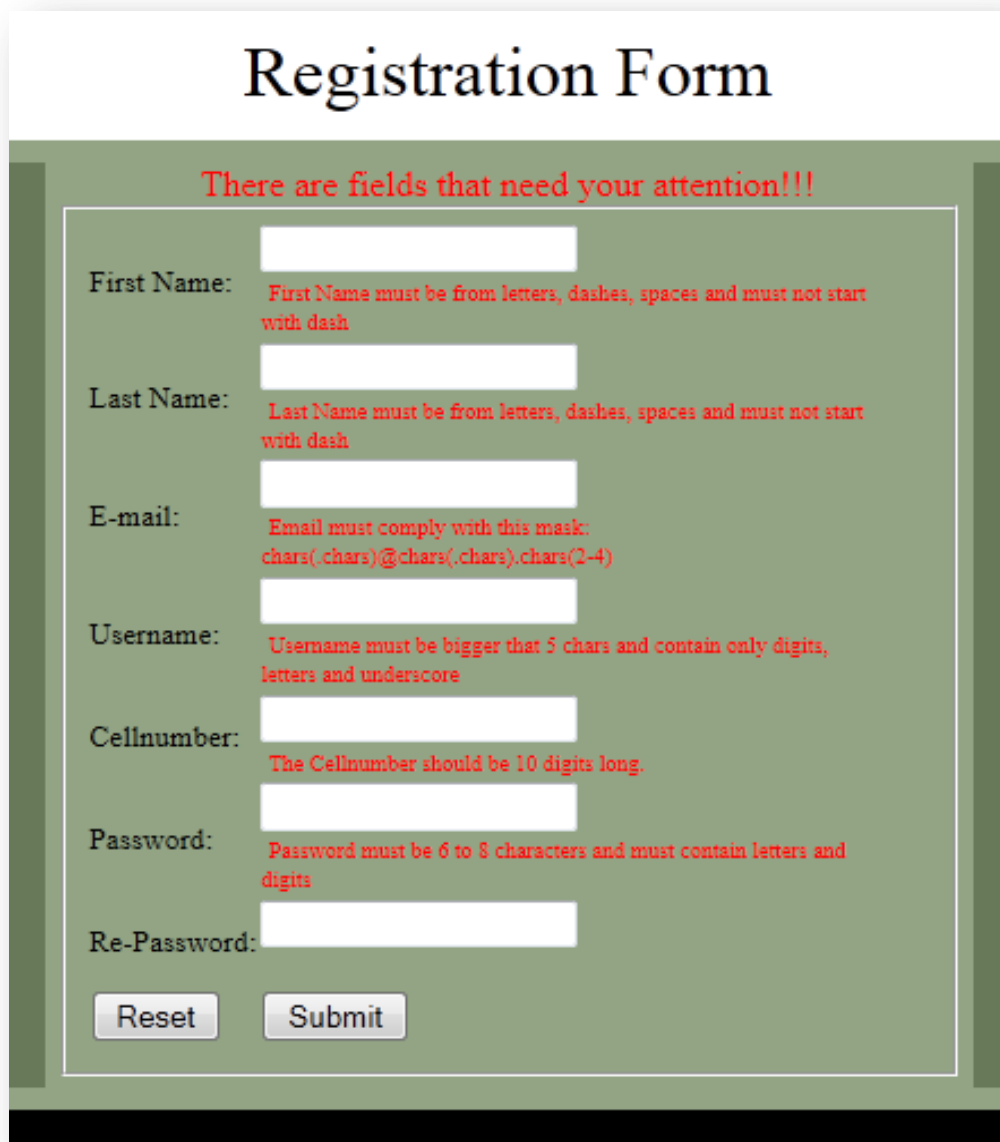
Figure 5.6: Change password UI

5.9. The registration interface

The registration page, *registrationForm.php*, offers users a simple interface through which they can create new accounts. Just like the login page, the registration page runs some routine checks when the user submits the page. The following is a list of checks that the registration page performs:

- The name and surname must consist only of letters.
- The email address must be written in the correct format (a@b.c).
- The email address must not be in use already.
- The username may only contain letters, digits and an underscore.
- The username must not be in use already.
- Cellphone number must be exactly 10 digits.

- The password must contain letters and digits and must be 6 to 8 characters long; this is for security reasons.
- When confirming the password, the second entry is compared to the first one to ensure that the same password has been entered twice.



The image shows a registration form titled "Registration Form" with a green background. At the top, a red warning message reads "There are fields that need your attention!!!". Below this, there are seven input fields, each with a red validation message:

- First Name:** First Name must be from letters, dashes, spaces and must not start with dash
- Last Name:** Last Name must be from letters, dashes, spaces and must not start with dash
- E-mail:** Email must comply with this mask: chars(.chars)@chars(.chars).chars(2-4)
- Username:** Username must be bigger that 5 chars and contain only digits, letters and underscore
- Cellnumber:** The Cellnumber should be 10 digits long.
- Password:** Password must be 6 to 8 characters and must contain letters and digits
- Re-Password:** (No message shown)

At the bottom of the form, there are two buttons: "Reset" and "Submit".

Figure 5.7: Validation of the Registration form

Upon successful submission, all the details supplied are entered and stored into a database, called *users*, which is responsible for keeping users information. The user is redirected to a page that thanks him/her for registering and is then given an option to login.

5.10. The welcome page

This page is displayed upon successful login. It does not do much except display a welcome message and provides links to the three of the four Web Services; namely: SMS Web Service, MMS Web Service and Email Web Service.

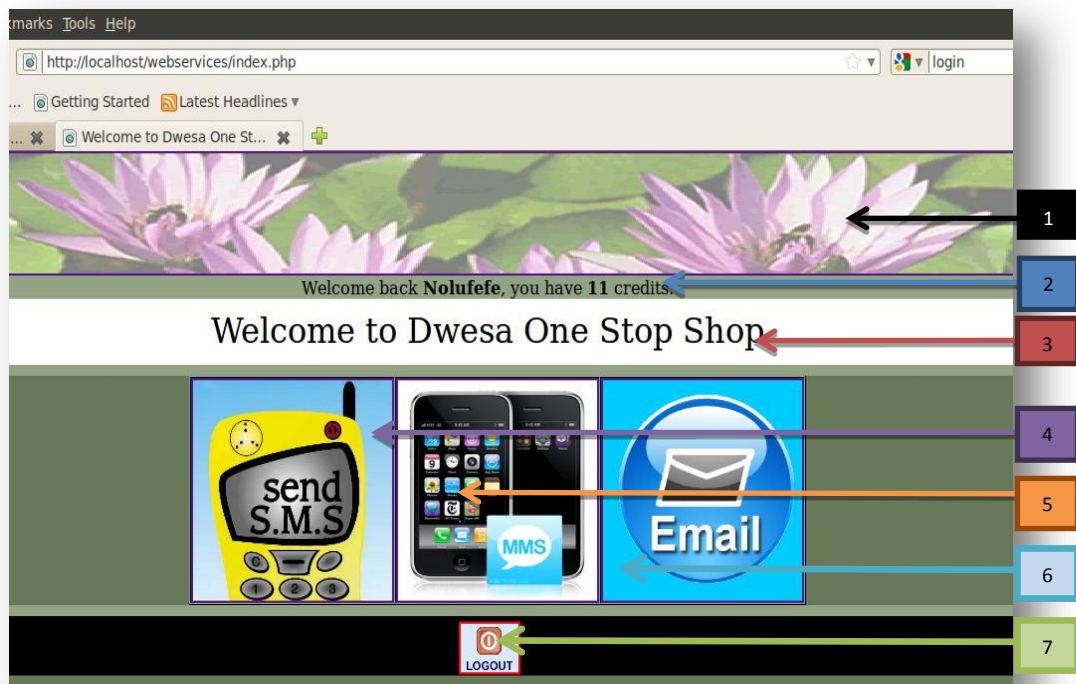


Figure 5.8: Welcome page

The numbers in the figure above are explained below:

On mouse click:

1: sends the user back to the welcome page displayed above. This image appears on every page of the one-stop-shop.

2: Displays the user's first name and credits.

3: Displays the title/name of the page.

4: Displays the SMS Web Service interface.

5: Displays the MMS Web Service interface.

6: Displays the Email Web Service interface.

7: This *logout* button appears in the footer section of all pages except for the *login*, *logout*, *registration*, *change password* and *recover password* pages. This is made possible through the use of the PHP `session_start()` function. In fact, this is the same function used in the *auth.php* to determine whether the user is logged in or not.

5.11. The logout page

When the user clicks on the *logout* button, the session that started when the user logged in is terminated and the user is redirected to a page that confirms that he/she has successfully logged out and he/she is also given an option to login again (Re-Login). The logout button will no longer be available in the footer section. After this operation, if the user clicks on the back button of his/her browser, he/she will be redirected to the login page.

```
7 <?php
8 if (!isset($_SESSION['logged'])) {
9     $url = 'http://' . $_SERVER
10     ['HTTP_HOST'] . dirname($_SERVER
11     ['PHP_SELF']);
12     if ((substr($url, -1) == '/') OR
13     (substr($url, -1) == '\\')) {
14         $url = substr ($url, 0, -1);
15     }
16     $url .= '/index.php';
17     redirect_to("{ $url }");
18 } else {
19     unset($_SESSION['logged']);
20     session_destroy();
21     redirect_to("Logged_out.php");
22 }
23 ?>
```

Listing 5.16 *logout.php*

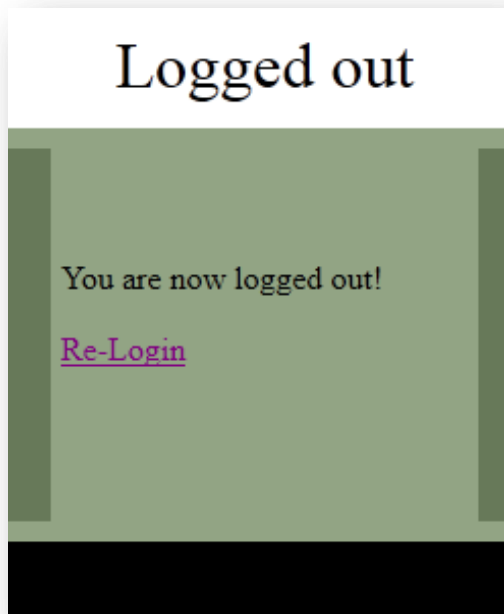


Figure 5.9: the user has successfully logged out.

5.12. SMS Web Service UI

This UI allows the user to send out an SMS. Each field is explained below the image.

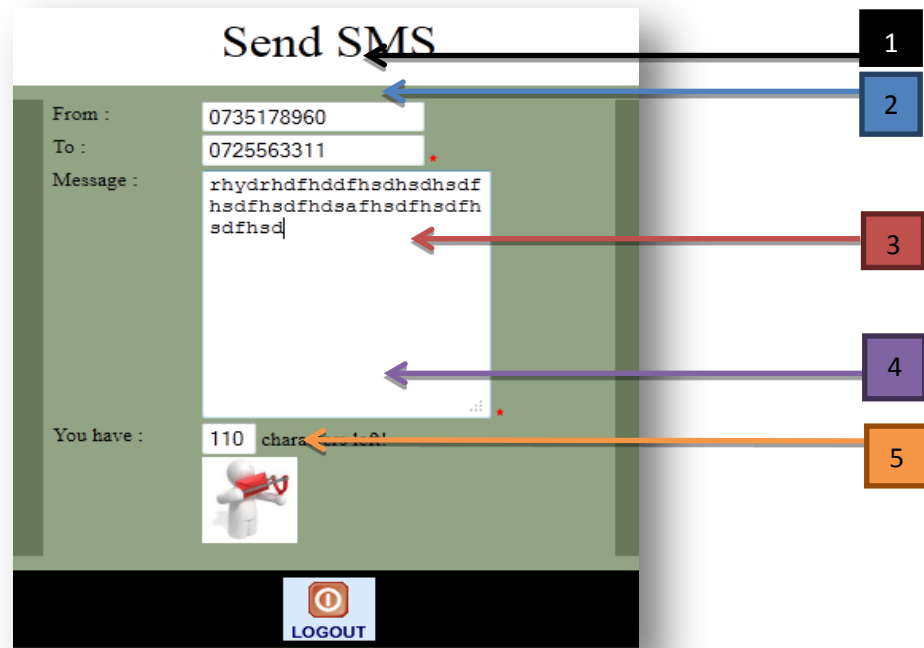


Figure 5.10: SMS User Interface

1: The number displayed in this text field is automatically filled in whenever the user opens the above page. Kannel allows the sender to send his/her cellular number to the recipient if he/she wishes to.

2: The number of the recipient must be filled in this field. It is mandatory, or else the execution will be terminated with an error message that reads “please insert recipient’s number!”

3: This is the field in which the message to send is entered. The form will not be submitted if this field is empty and a JavaScript code is attached to it with the purpose of limiting the message to only 160 characters.

4: This field displays the number of characters the user still has left.

5: this is the send button. Once this button is pressed, the following takes place:

- The required fields are checked:
 - The *To* field is checked to verify whether it is empty or not and, if it is not empty, the number submitted must be digits only and not longer than 10

characters. If the field is empty, or the number submitted does not contain digits only, or it is not 10 characters long, the execution is terminated with an error message.

- The *message* field is also checked to make sure that it is not empty. The user is not allowed to send empty messages.
- Upon successful submission, a request is sent to the SOAP-server, via a SOAP-client named *sms.php*, to consume the SMS Web Service.

These fields are easy to understand and for this reason the UIs in the next sections are not going to be discussed in detail. The focus will only be on new features and fields.

5.13. MMS Web Service UI

This UI allows the user to send out an MMS.

The screenshot shows a web form titled "Send MMS". The form has a green background and contains the following elements:

- Fields for "From:", "To:", and "Subject:".
- A large text area for "Message:".
- A character count: "You have : 160 characters left!".
- Fields for "Image:" and "Sound:" with "Browse..." buttons.
- A "LOGOUT" button at the bottom.

Two arrows point to specific parts of the form: a purple arrow labeled "1" points to the "Message:" field, and a red arrow labeled "2" points to the character count.

Figure 5.11: MMS User Interface

The MMS Web Service UI looks more like the one discussed in the previous section except for the two new fields that have been added. At least one of these two fields must contain data when the form is submitted. Should this not be the case, the application stops executing and returns an error message notifying the sender that he/she must upload either an image file or a sound file in order to consume the service.

When the page is submitted, the following takes place.

- When an image is uploaded, at least two things are verified when the form is submitted:
 - The type of the image: only images with the following extensions are accepted: *.jpeg*, *.jpg* and *.png*
 - The size of the image: any image bigger than 1MB is not uploaded.
- The same validation is done for sound files.
 - The type of sound file: only sound files with the following extensions are accepted: *.mid*, *.midi* and *.mp3*.
 - The size of the sound file: sound files with *.mid/.midi* extensions are generally not big (less than 1MB). When a user wants to send an mp3 sound file, he/she must make sure that the size of the file is less than 1MB.
- Upon successful submission of the page, the request is sent to the SOAP-Server via a SOAP-client named *mms.php*.

5.14. EMAIL Web Service UI

This UI allows users to send out emails.

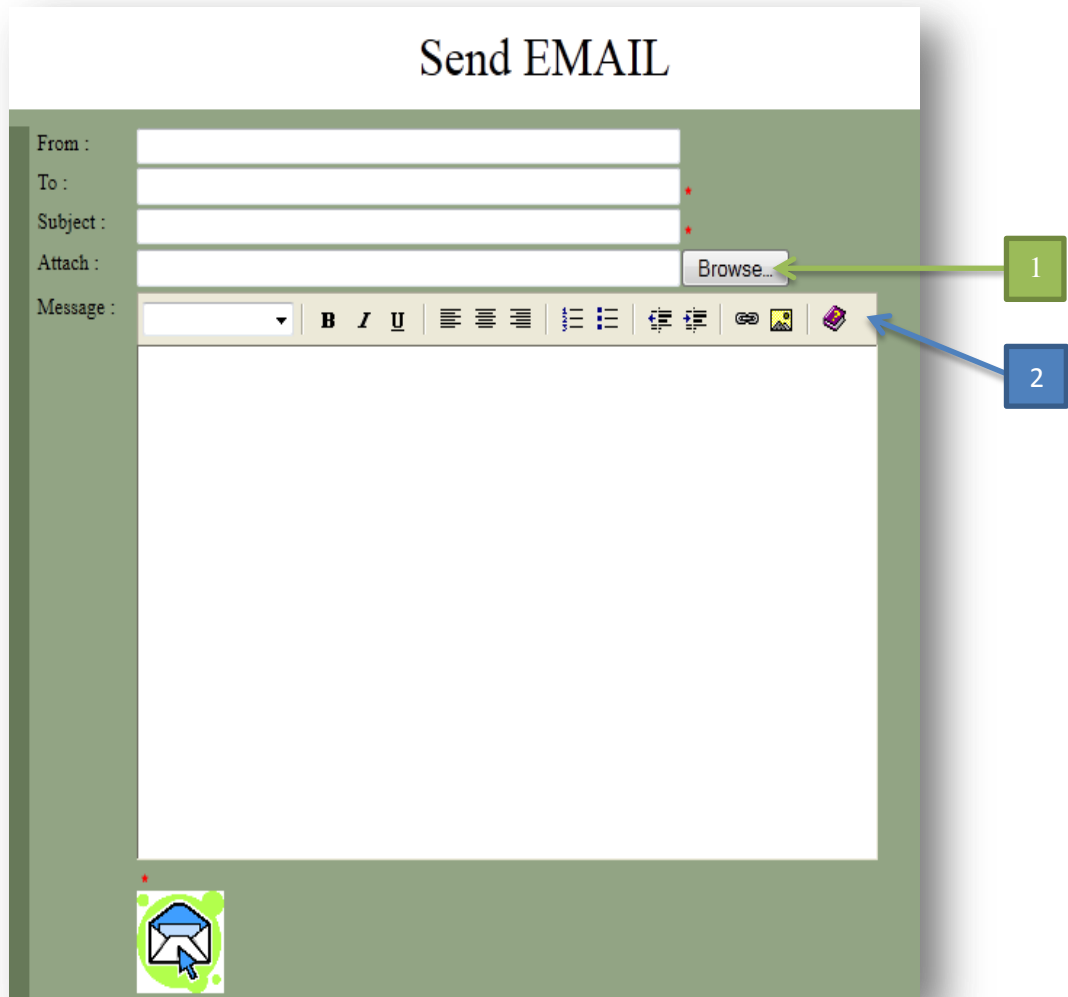


Figure 5.12: Email User Interface

There are two items to consider from this UI.

- Image 1: This field allows the user to send emails with attachments.
- Image 2: This is a WYSIWYG application that allows the user to format the text according to his/her needs.

When this page is submitted the following takes place:

- The *To* field is checked to see whether it is empty and, if it is not empty, the content is checked to verify whether it conforms to an email format. If the field is empty or the content is not a proper email address, the execution is terminated and an error message is displayed on the user's screen.

- The *Message* field is also checked to ascertain whether the field is empty. If it is empty, the execution is terminated and an error message is displayed on the user's screen.
- Upon successful submission, the request is sent to the SOAP-server via a SOAP-client called *email.php*.

5.15. IM Web Service UI

This UI is only available to the system administrator. It allows the system's administrator to send out IM notifications to system users.



Figure 5.13: IM User Interface

This UI is simple to understand. It is mostly used to send out IM notifications to the system XMPP server. Users receive these IM notifications when they are online.

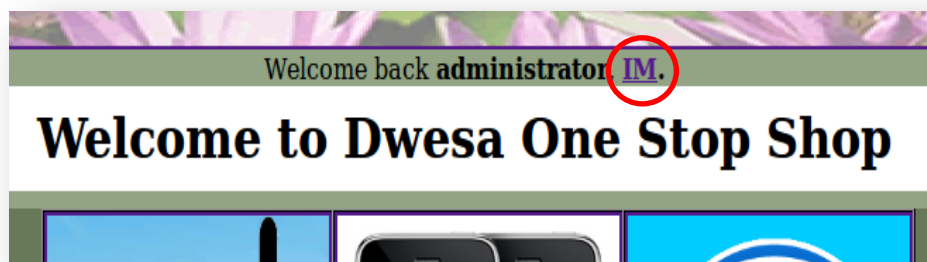


Figure 5.14: Link to the IM UI

5.16. Conclusion

This chapter discusses the implementation of the system's components. The next chapter discusses the system's testing and results.

CHAPTER

VI

6. TESTING AND RESULTS

6.1. Introduction

The previous chapter discussed how each component of the system has been implemented and how they relate to each other. This chapter focuses on a number of tests that the system goes through and their results.

6.2. Testing of back-end Applications

There are a number of applications that are running in the system's background that need to be tested in order to verify whether they have been configured properly.

6.2.1. Kannel

Kannel is used by the SMS Web Service as a SMS gateway and by the MMS Web Service as a WAP gateway. Consequently, the *bearerbox*, *smsbox* and *wapbox* need to be tested to verify whether they are working properly.

```
2010-11-22 11:20:23 [3456] [6] DEBUG: AT2[/dev/ttyS0]: <-- OK
2010-11-22 11:20:23 [3456] [6] INFO: AT2[/dev/ttyS0]: Phase 2+ is supported
2010-11-22 11:20:23 [3456] [6] DEBUG: AT2[/dev/ttyS0]: --> AT+CSMS=1^M
2010-11-22 11:20:23 [3456] [6] DEBUG: AT2[/dev/ttyS0]: <-- +CSMS: 1,1,1
2010-11-22 11:20:23 [3456] [6] DEBUG: AT2[/dev/ttyS0]: <-- OK
2010-11-22 11:20:23 [3456] [6] DEBUG: AT2[/dev/ttyS0]: --> AT+CNMI=1,2,0,1,0^M
2010-11-22 11:20:23 [3456] [6] DEBUG: AT2[/dev/ttyS0]: <-- OK
2010-11-22 11:20:23 [3456] [6] INFO: AT2[/dev/ttyS0]: AT SMSC successfully opened.
```

Figure 6.1: *bearerbox* at work.

The last line in the image above shows that the *bearerbox* has successfully established a connection with the SMSC. This is an indication that Kannel has been configured properly.

```
2010-11-22 11:21:56 [3498] [0] DEBUG: Started thread 6 (gw/smsbox.c:http_queue_thread)
2010-11-22 11:21:56 [3498] [6] DEBUG: Thread 6 (gw/smsbox.c:http_queue_thread) maps to pid 3498.
2010-11-22 11:21:56 [3498] [0] INFO: Connected to bearerbox at localhost port 13001.
2010-11-22 11:21:56 [3498] [0] DEBUG: Started thread 7 (gw/heartbeat.c:heartbeat_thread)
2010-11-22 11:21:56 [3498] [7] DEBUG: Thread 7 (gw/heartbeat.c:heartbeat_thread) maps to pid 3498.
```

Figure 6.2: *smsbox* at work.

The third line from the bottom shows that the *smsbox* is successfully connected to the *bearerbox*. This is an indication that Kannel is now ready to send out and receive SMSs

6.2.2. Mbuni

Mbuni is configured to run as a VAS gateway in this project. The only test that needs to be done is to verify whether the *mmsbox* runs properly. This means that the *mmsbox.conf* has been configured properly.

```
root@dwesaproject:/usr/local/bin# mmsbox -v 0 /etc/mbuni/mmsbox.conf
2010-11-30 10:33:04 [3661] [0] INFO: Debug_lvl = 0, log_file = <none>, log_lvl = 0
2010-11-30 10:33:04 [3661] [0] INFO: -----
2010-11-30 10:33:04 [3661] [0] INFO: Mbuni MMSBox version 1.4.0 starting
2010-11-30 10:33:04 [3661] [0] INFO: Added logfile '/var/log/mmsbox.log' with level '0'.
2010-11-30 10:33:04 [3661] [0] INFO: Started access logfile '/var/log/mmsbox-access.log'.
2010-11-30 10:33:04 [3661] [0] INFO: HTTP: Opening server at port 10001.
2010-11-30 10:33:04 [3661] [0] DEBUG: Started thread 1 (glib/fdset.c:poller)
```

Figure 6.3: *mmsbox* at work

The image above shows that Mbuni's *mmsbox* is running properly and is ready to send out MMSs.

6.2.3. Postfix

Postfix is used as the MTA. In order to test whether it has been configured properly, an email is sent to one of the users created on localhost.

```
1291111557.Vfb01I3a...052.dwesaproject.com ✕
Return-Path: <jimmy@dwesaproject.com>
X-Original-To: jimmy@dwesaproject.com
Delivered-To: jimmy@dwesaproject.com
Received: from dwesaproject.com (dwesaproject.com [172.20.56.68])
        by dwesaproject.com (Postfix) with ESMTP id 61B67120273
        for <jimmy@dwesaproject.com>; Tue, 30 Nov 2010 12:04:19 +0200 (SAST)
Message-Id: <20101130100526.61B67120273@mail.dwesaproject.com>
Date: Tue, 30 Nov 2010 12:04:19 +0200 (SAST)
From: jimmy@dwesaproject.com
To: undisclosed-recipients:;

Test from remote host
Test #2
```

Figure 6.4: User name *Jimmy* has received the test email.

6.3. Non-functional testing

This section focuses on the non-functional testing listed in section 4.2.2. Further non-functional testing can be viewed in Appendix E (compatibility testing).

6.3.1. Compatibility

The purpose of this testing is to determine whether this system performs according to requirements when run in an environment different from the original.

This entire system is developed and implemented on a Linux machine. This system does run well on a Windows machine with the exception of Kannel and Mbuni. The specific challenges associated with running Kannel and Mbuni on a Windows machine are discussed in section 7.3.2.

The image below shows that this system's SOAP-server works properly with clients from different programming languages. This also shows that other machines/systems can successfully plug in to this system's SOAP-server and request to consume its Web Services.

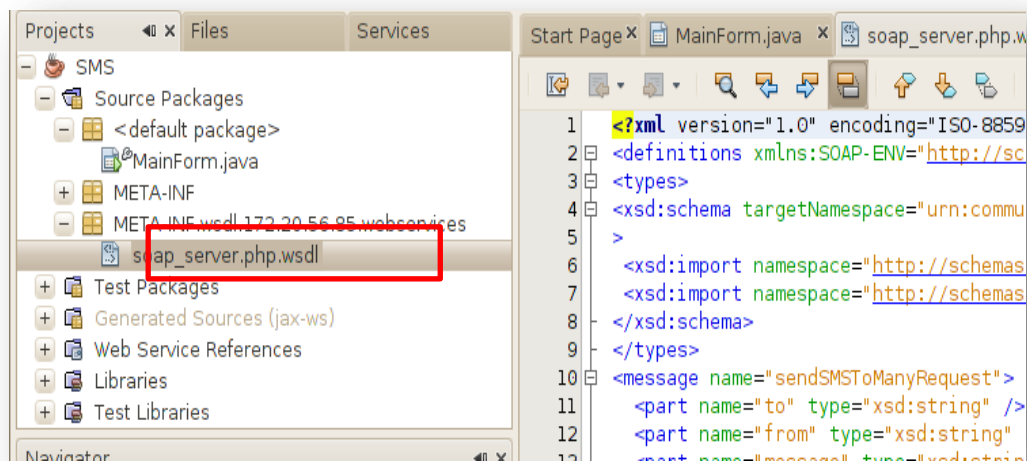


Figure 6.5: A java application communicating with the SOAP-server.

6.3.2. System Usability Scale.

System Usability Scale (SUS) is a simple, ten-item attitude Likert scale giving a global view of subjective assessments of usability (Brooke, 1996).

The usability of a system, as defined by the ISO standard ISO 9241 Part 11, can only be measured by taking into account the context of use of the system (Abran, et al. 2003). The focus is on who is using the system, what they are using it for, and the environment in which they are using it (Abran, et al. 2003).

Measurements of usability have several different attributes (Abran, et al. 2003):

- Effectiveness: Can users successfully achieve their objectives?
- Efficiency: How much effort and resource is expended in achieving those objectives?
- Satisfaction: Was the experience satisfactory?

A questionnaire is used in order to do this system usability testing. The first part of the questionnaire focuses on the user’s background as far as communication technologies, such as mobile phone services and email, are concerned. The second part of the questionnaire focuses on SUS.

6.3.2.1. Part one: User’s background

Question	Yes	No
Are you computer literate?,		
Do you have access to the Internet?		
Do you have an email address?		
Do you own a cell phone?		

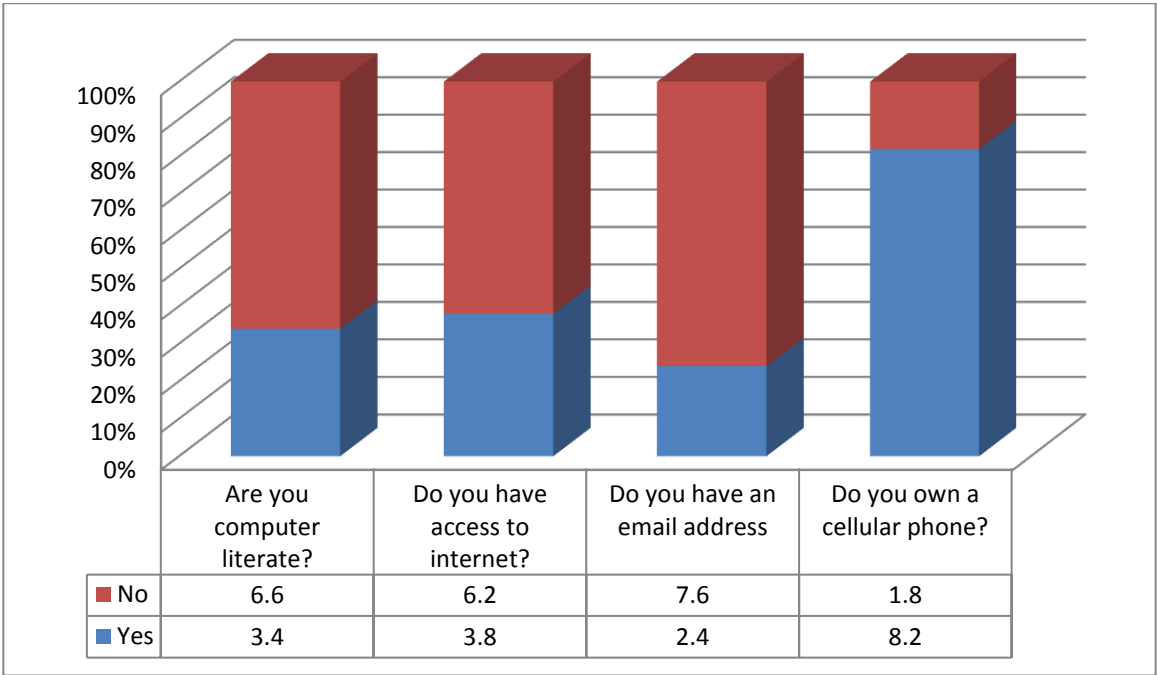
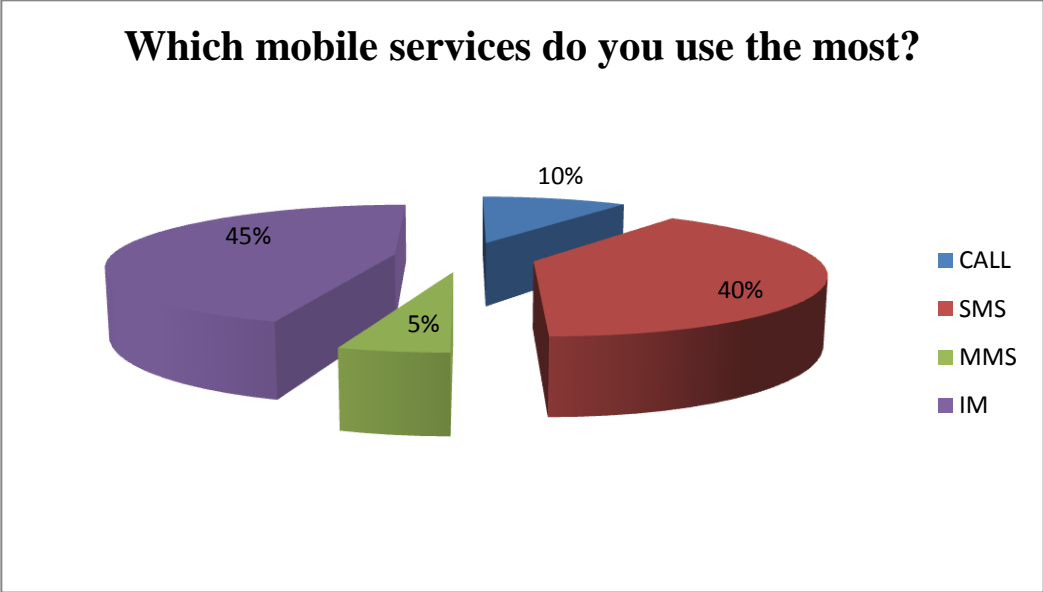


Figure 6.6: User's background

The graph above shows that the majority of people involved in this system usability test are not computer literate, they do not have access to the Internet and they do not have email addresses. But most of them own cellular phones.

Which mobile services do you use the most?							
CALL		SMS		MMS		IM	



The statistics above show that people involved in this system usability test mostly use the IM and SMS technologies for communication purposes.

The IM technology is frequently used because of a mobile application called *Mxit*. This application offers a social forum and IM services to its consumers at a very low cost. Other technologies that have contributed to the rise of IM technologies in mobile phones are Facebook and Twitter.

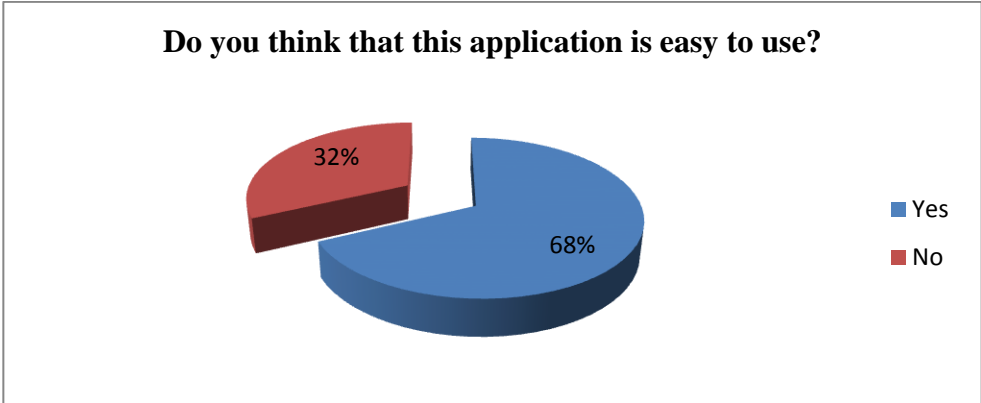
The SMS technology is also used quite often because of its low cost, especially during off-peak periods. The other factor that has contributed to an increase in the use of SMS is the *Please Call Me* service. This service is free and people use it often to send out very short messages.

Calls are expensive during peak times, during the day, and are often made to people who are far away. MMS technologies are not often used because most people involved in this survey use GSM mobile phones.

6.3.2.2. Part two: SUS

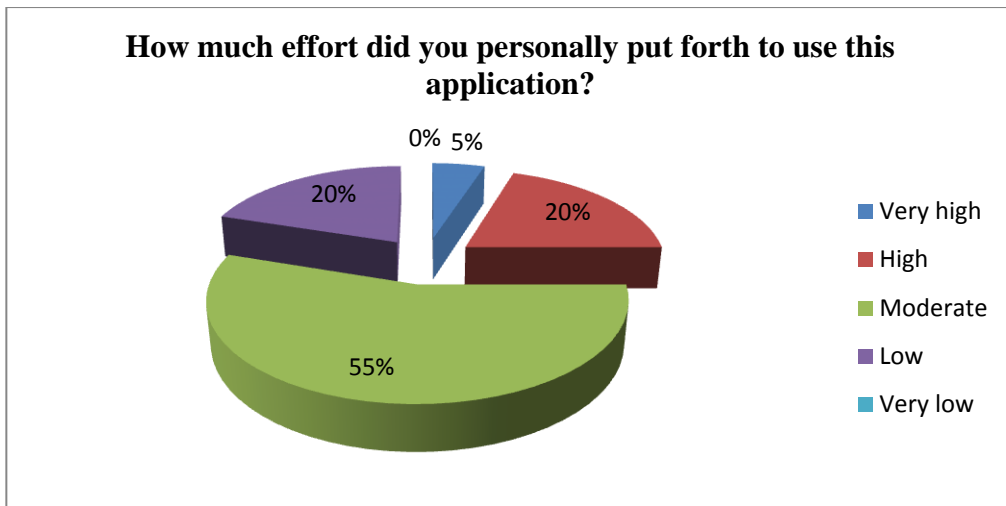
1. Do you think that this application is easy to use?				Yes	No
2. How much effort did you personally put forth to use the application?	Very high	High	Moderate	Low	Very Low
3. How easy is it to understand the user interface?	Very difficult	Difficult	Moderate	Easy	Very Easy
4. How do you rate the entire system?	Needs improvement	Adequate	Good	Very good	Excellent

- Question 1: Do you think that this application is easy to use?



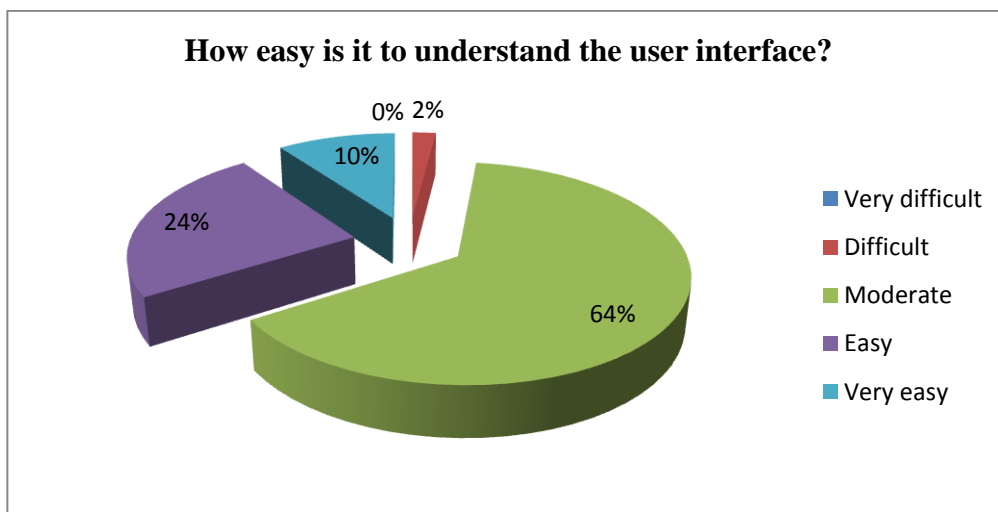
The pie chart above shows that even though 66% of the people involved in this survey are not computer literate (Figure 6.6: User's background), 68% think that the system is easy to use.

- Question 2: How much effort did you personally put forth to use the application?



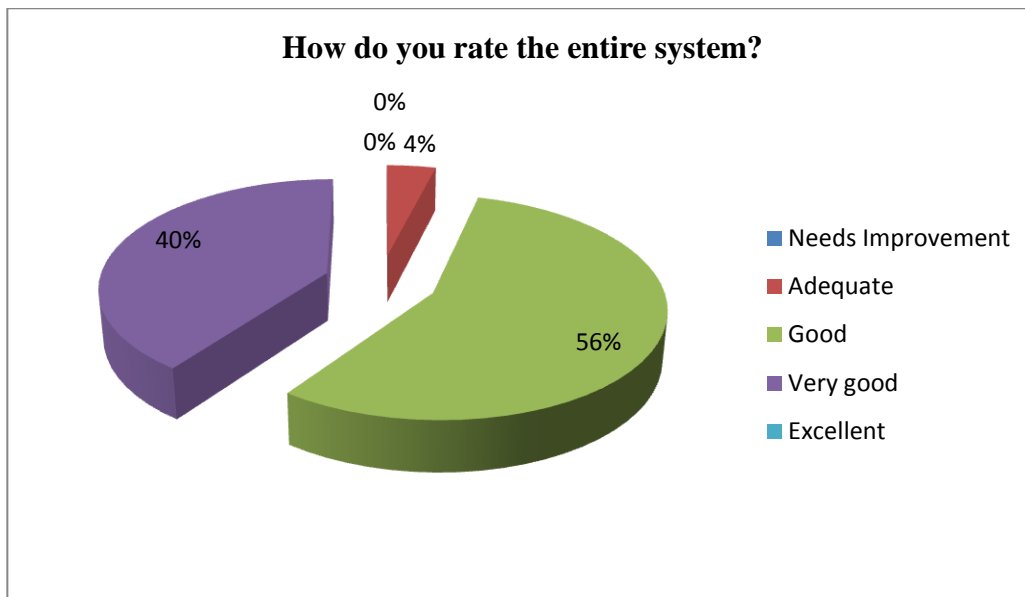
The pie chart above shows that more than half of the people involved in this did not have to put in much effort to perform tasks on the system. One of the reasons for this is that the UI is simple and easy to use. This is shown in the next question.

- Question 3: How easy is it to understand the user interface?



The pie chart above shows that more than half of the people involved in the survey find the UI easy to use. This is due to the fact that the Web Services UI uses familiar icons that speak for themselves.

- Question 4: How do you rate the entire system?



The pie chart above shows that there is no suggestion for improvement of the system. The system is mostly rated to be good.

6.4.Functional Testing

Below is a list of testing conducted to verify whether the user can successfully interact with the system's front-end component.

6.4.1. Registration

This component is tested to verify whether a user can successfully create a new account and his/her details are successfully stored in the users' database.

Registration Form

Personal Details:
All fields are required.

First Name:

Last Name:

E-mail:

Username:

Cellnumber:

Password:

Re-Password:

Acknowledgement

Thank you **User LastName** for registering.

You have **0** credits in you account. Purchase credits from the service administrator in order to be able to make use of the services offered.

[Login](#)

Figure 6.7: User Registering

user_id	fname	lname	email	username	cellnumber	password	credit
1	User	LastName	user1@dwesaproject.com	user123	0715628964	5a4687d87dbfaeb5a2a698f3ec80a4b2	0

Figure 6.8: Registration of User successful

Figure 6.8 shows a user creating a new account by filling in the registration form and the acknowledgement message displayed upon successful registration. Figure 6.8 shows the new user's details which are successfully stored in the database.

6.4.2. Login

This component is tested to verify whether a user can successfully create a new account and the details are successfully stored in the users' database.



Figure 6.9: User logging in

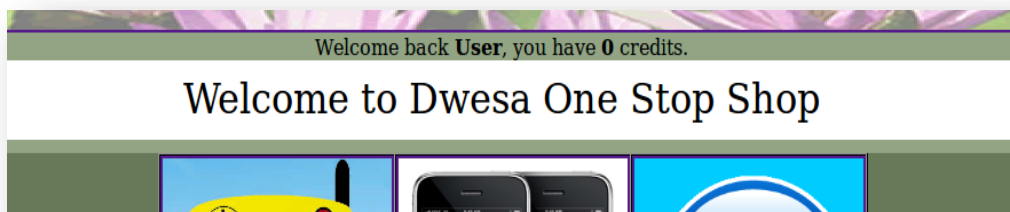


Figure 6.10: User successfully logged in

Figure 6.9 shows a user signing in by supplying either a username or cell number and password. Figure 6.10 shows that the user has successfully logged into the system.

6.4.3. Change Password

This component is tested to verify whether a user can successfully change his/her password by replacing the old password with a new one in the users' database.

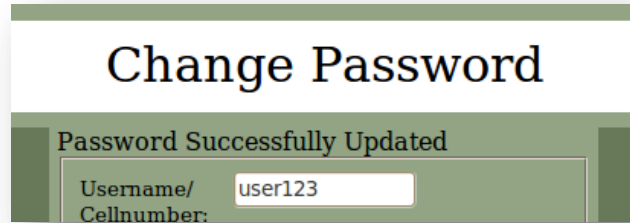


Figure 6.11: User gets a confirmation on the UI.

```
email | username | cellnumber | password
-----|-----|-----|-----
user1@dwesaproject.com | user123 | 0715628964 | c15d309c76c545bccff133852eebbd07
email | username | cellnumber | password
-----|-----|-----|-----
user1@dwesaproject.com | user123 | 0715628964 | 5a4687d87dbfaeb5a2a698f3ec80a4b2
```

Figure 6.12: Password successfully changed in the users' database.

Figure 6.11 shows that the user has successfully submitted the request to change password. Figure 6.12 shows that the new password is successfully encrypted and it is successfully entered in the database replacing the user's old password.

6.4.4. Recover Password

This component is tested to verify whether the system can successfully create a temporary password, upon the user's request, update the user's password in the users' database and send the temporary password to the user's registered email address. The *recover password* function uses a SOAP-client in order to request the email Web Service from the SOAP-server.

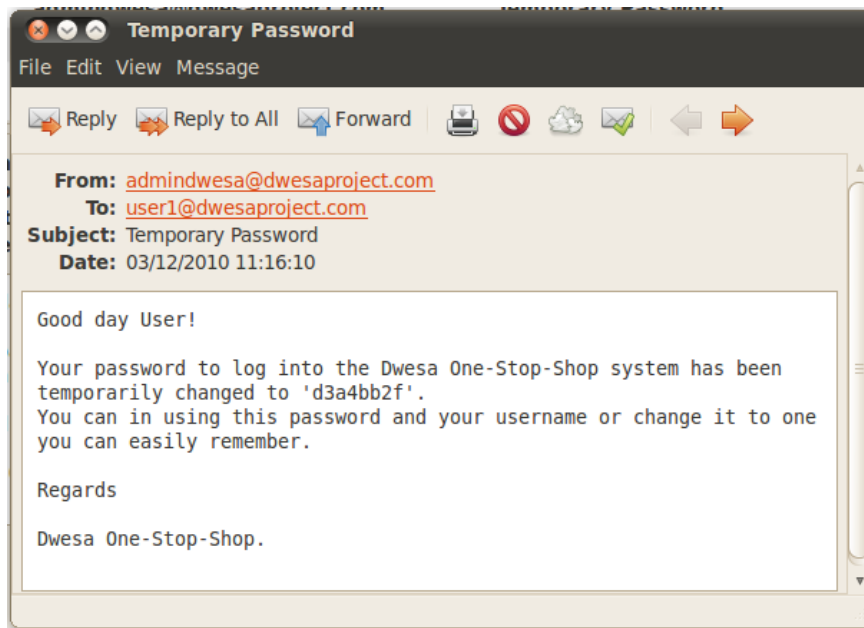


Figure 6.13: *User has successfully received a temporary password via email.*

This test does not only show that a user can successfully recover a password, but it also shows that the system successfully establishes a machine-to-person type of communication. Figure 6.13 shows that the user has successfully received an email with the temporary password.

6.5. Conclusion

This chapter focuses on different techniques used to test different components of the system. The next chapter concludes the study.

CHAPTER

VII

7. DISCUSSION AND CONCLUSION

7.1. Introduction

The previous chapter focuses on the different techniques used to test the system. This chapter summarizes the work undertaken in this project. It lists the achievements and discusses the challenges faced during the development and implementation of the system. Before giving an overall conclusion, it outlines suggestions of further research.

7.2. Achievements

A comprehensive literature review has been conducted in order to build this system. This culminated in defining the system objectives and methodology. In addition, it led to defining the technologies required which, in turn, helped to design the system architecture. Parallel to the system architecture, a prototype has been implemented in order to test and validate the architecture.

The design of the architecture is based on the SOA approach which focuses on exposing a service in the server-client environment. This allows more than one system/department to make use of the same service. It helps reduce the duplication of the service and, as a result, it helps save time and resources.

To demonstrate the feasibility and validity of the proposed design solution, a system prototype was implemented and tested. This prototype is designed to take advantage of the existing ICTs in the research area. This prototype features a novel communication engine which exposes a number of SOAP-based Web Services that allows machine-to-machine, machine-to-person and person-to-person types of communication in the SLL and within the Dwesa community.

From the implementation perspective, the system satisfies the objectives of the project as described in section 1.4.

A SOA system which offers its functionalities as Web Services is implemented and other application software that forms part of the SLL middleware framework successfully establishes machine-to-machine communication. For instance, the eHealth system successfully connects to this system's SOAP-server and makes use of the Email Web Service. This meets the first objective of the research.

A user interface is successfully added to the system to allow users from Dwesa to make use of the system Web Services for communication purposes. This meets the second objective of the research.

This research did, in fact, encounter a few challenges which are discussed in the next section.

7.3. Challenges and limitations

This section discusses the challenges and limitations encountered during the development and implementation of the system.

7.3.1. Challenges

There were a few challenges encountered when developing and deploying the MMS Web Services.

1. The MMS gateway: It took long to actually find an appropriate MMS gateway for the system. The challenge came from the fact that most MMS gateways that were accessible were mostly designed for Windows OS and not compatible with Linux OS.
2. MMS binary file: MMSs are more complex than SMSs. They involve the sending of text, images and sound files. The MMS gateway, in use, only sends out binary files. This led to the need to encode MMS messages into binary files. The challenge with this was that it took long to find a compatible application or library for encoding text, images and sound files into binary files.

7.3.2. Limitations

Kannel and Mbuni are designed to run on a Linux machine only. Due to that, the system, as a whole, is limited to a Linux machine.

7.4. Future work

The most obvious future work is the implementation of a more secure SOAP-server. At the moment, users only need authentication and authorization when consuming Web Services from the user interface. But, from a machine-to-machine communication point of view, the SOAP-server is freely accessible.

Another future work is the implementation of a 3G application that allows video streaming through the use of Web Services.

7.5. Overall conclusion

This study has described the design of SOA wrappers for communication purposes at a machine-to-machine, machine-to-person and person-to-person level. Most importantly, this thesis outlines a system which uses modern ICTs and modern ways of communication to provide a pool of SOAP-based Web Services which can be used in the Siyakhula Living Lab and by the Dwesa community.

REFERENCES

8. REFERENCES

Abran, A., Khelifi, A., Suryan, W., & Seffah, A. (2003). Usability Meanings and Interpretations in ISO Standards. *Software Quality Journal*, 11(4):325-338.

Alcatel-Lucent. (2006). XML Web Services & API's. Available from: <http://enterprise.alcatel-lucent.com/?product=XMLServices&page=overview> (Accessed 12 October 2010).

Bodic, G. L. (2005). *Mobile messaging technologies and services SMS, EMS and MMS*. West Sussex: John Wiley & Sons Ltd.

Brooke, J. (1996). SUS: A Quick and Dirty Usability Scale. In: P.W. Jordan, B. Thomas, B.A. Weerdmeester & I.L. McClelland (Eds.), *Usability Evaluation in Industry*. London: Taylor & Francis.

Burns, C. (2010). Mobile Development Projects: A Sampling. Available from: http://pdf.usaid.gov/pdf_docs/PNADS558.pdf (Accessed 06 December 2010).

Coetsee, F. (2010). Why Social Media in South Africa will NOT fail. Available from: <http://www.socialmediastrategy.co.za/index.php/general/why-social-media-in-south-africa-will-not-fail/> (Accessed 22 November 2010).

Craydon, M. (2004). MMSlib: Encode and Decode MMSes with PHP. Available from: <http://www.postneo.com/2004/08/06/mmslib-encode-and-decode-mmses-with-php> (Accessed 25 January 2010).

Crugnalo, A. (2006). Using Web service With Flash and Nusoap. Available from: <http://www.sephiroth.it/tutorials/flashPHP/webService/> (Accessed 15 April 2010).

Curtain, R. (2004). Information and communications technologies and development: Help or hindrance. *Australian Agency for International Development*. Available from: <http://www.Curtain-Consulting.net.au> (Accessed 20 October 2010).

Dalvit, L., Muyingi, H., Terzoli, A. & Thinyane, M. (2007). The Deployment of an e-Commerce Platform and Related Projects in a Rural Area in South Africa. In: Proceedings of the International Journal of Computing and ICT Research, 1(1):3.

Eriksson, M., Niitamo, V.-P., Kulkki, S. & Hribernik, K. A. (2006): State of the Art and Good Practice in the Field of Living Labs. In: Proceedings of the 12th International Conference on Concurrent Enterprising: Innovative Products and Services through Collaborative Networks. Italy: Milan.

Fink, A., Rodrigues, B., Tolj, S., Syvänen, A., Malysh, A., Wirzenius, L. & Marjola, K. (2010). Kannel svn-r4865 User's Guide: Open Source WAP and SMS gateway. Available from: <http://www.kannel.org/download/kannel-userguide-snapshot/userguide.pdf>. (Accessed 29 April 2010).

Flickenger, R. (2008) (2nd ed.). *Wireless networking in the Developing World: A practical guide to planning and building low-cost telecommunications infrastructure*. Seattle: Hacker Friendly LLC.

Goldsmith, A. (2005). *Wireless Communications*. California: Stanford University.

Hahn, R. (2008). All things Africa and ICT. Available from:<http://psdblog.worldbank.org/psdblog/2008/08/all-things-afri.html> (Accessed 06 December 2010).

Heeks, R. (2009). The ICT4D 2.0 Manifesto: where next for ICTs and international development. Development Informatics, Working Paper Series. Paper No 42. Development Informatics Group. Manchester University, Institute for Development Policy and Management.

Helsinki Living Lab. (2007). What is a Living Lab? Available from: <http://www.helsinkilivinglab.fi/node/162> (Accessed 20 July 2010).

Iskold, A. (2007). Evolution of Communication: From Email to Twitter and Beyond. Available from: http://www.readwriteweb.com/archives/evolution_of_communication.php. (Accessed 20 November 2010).

Jakachira, B.T., 2009. Implementing an integrated e-Government functionality for a marginalized community in the Eastern Cape, South Africa. Masters dissertation. Alice: University of Fort Hare.

Josuttis, M. N. (2007). *In SOA in practice: The art of distributed system design*. California: O'Reilly.

Kaplan, A.M. & Haenlein, M. (2010). Users of the world, unite! The challenges and opportunities of Social Media. *Business Horizons*. 53(1):59-68.

Kumar, P., Perreira, M., Vaidya, P., Vosseler, F. & Peltz, C. (2006). Moving from point-to-point integrations to SOA-based integrations. *Hewlett-Packard Technical Information*.

LaFraniera, S. (2010). Expansion of the Cell Phone Network in Rural Africa. Available from: <http://www.castlelab.princeton.edu/EnergyResources/2011/Wong-Prospectus.pdf> (Accessed 07 December 2010).

Living Labs in Southern Africa. Overview. Available from: <http://llisa.meraka.org.za/index.php/Overview> (Accessed 20 July 2010).

lyingonthecovers.net (2006). Getting complex with PHP and NuSOAP (Part 1). Available from: <http://www.lyingonthecovers.net/?p=39> (Accessed 03 November 2010).

Mbuni. (2004). Free, Open Source MMS Gateway. Available from: <http://www.mbuni.org/userguide.shtml> (Accessed 29 April 2010).

Negus, C. (2007). *Linux 2007 Edition*. Indianapolis: Wiley Publishing, Inc.

Nichol, S. (2004). Programming with NuSOAP Using WSDL. Available from: <http://www.scottnichol.com/nusoapprogwsdl.htm> (Accessed 29 April 2010).

Oracle. (2008). Ensuring Web Service Quality for Service-Oriented Architectures. Available from: <http://www.oracle.com/technetwork/oem/grid-control/overview/wp-ensuring-1.pdf> (Accessed 22 July 2010).

Pade, C., Palmer, R., Kavhai, M. & Gumbo, S. (2009). Siyakhula Living Lab: Baseline Study Report, Unpublished report.

Postfix. (n.d.). Postfix Basic Configuration. Available from:
http://www.postfix.org/BASIC_CONFIGURATION_README.html (Accessed 22 April 2010).

Pun, R., Shields, R., Poudel, R. & Mucci, P. (2006). Nepal wireless networking project. Available from: <http://nepalwireless.net/images/stories/npw.pdf> (Accessed 05 December 2010).

Reitman, L., Ward, J. & Wilber, J. (2007). Service Oriented Architecture (SOA) and Specialized Messaging Patterns. Available from:
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.5751&rep=rep1&type=pdf> (Accessed 10 April 2010).

Richards, R. (2006). *Pro PHP, XML and Web Services*. California: Apress.

Sayo, P. (Editor). (2004). Globalization and WTO: ICT, Trade and Competitiveness. In *ICT Policies and e-Strategies in the Asia-Pacific*. New Delhi: Elsevier. Available from:
<http://www.apdip.net/publications/ict4d/e-strategies.pdf> (Accessed 23 July 2010).

Schoolnet Uganda. (2007). Uganda rural schools VSAT school-based Telecentre (SBT) project. Available from: <http://schoolnetuganda.sc.ug/projects/completed-projects/uganda-rural-schools-vsbt-project.htm> (Accessed 04 December 2010).

Scott, M. S. (2010). Investigation and Development of an e-Judiciary Service for a Citizen-Oriented Judiciary System in Rural Community. Masters dissertation. Alice: University of Fort Hare.

Scott, N., Batchelor, S., Ridley, J. & Jorgensen, B. (2004). The impact of mobile phones in Africa. *Commission for Africa, London*.

Smyth, G. (2005). Wireless Technologies Bridging the Digital Divide in Education. *International Journal of Emerging Technologies in Learning*. 1(1):2

Srinivasan, L. & Treadwell, J. (2005). An Overview of Service-oriented Architecture, Web Services and Grid Computing. *Hewlett-Packard Technical Information*. 2:2-6.

Thioune, R. M. (Editor). (2003). Opportunities and Challenges for Community Development. Information and Communication Technologies for Development in Africa. *Opportunities and Challenges for Community Development*. 1:12.

Timmermans, H. G. (2004). Rural livelihoods at dwesa/cwebe: poverty, development and natural resource use on the wild coast. Masters dissertation. Grahamstown: Rhodes University.

Viale, R. & Ghiglione, B. (1998). The Triple Helix model: a Tool for the Study of European Regional Socio Economic Systems. *The IPTS Report*. 29:87

Vugt, S. V. (2008). *Beginning Ubuntu Server Administration. From Novice to Professional*. California: Apress.

Wandschneider, M. (2007). Setting up, Configuring, and Using Kannel to send/receive SMS messages. Available from: <http://Setting-up-Configuring-and-Using-13@.htm> (Accessed 26 November 2009).

Wassermann, T. (2009). Jabber with PHP – Part 1 (XMPPHP). Available from: <http://www.brownphp.com/2009/03/jabber-with-php-part-i-xmpphp/> (Accessed 10 July 2010).

Werdmuller, B. (2010). Build a web-based notification tool with XMPP. Available from: <http://www.ibm.com/developerworks/xml/tutorials/x-realtimeXMPPtut/section4.html> (Accessed 10 September 2010).

Wertlen, R. (2010). *A Design of a Middleware Solution for Connected Rural Digital Access Nodes Enabling a Multitude of Applications*. Unpublished thesis. Alice: University of Fort Hare.

APPENDICES

9. APPENDIX A – Basic Technologies Required

9.1. Operating system

This system can be developed on either Windows OS, or MAC OS, or Linux OS. But due to the fact that this project is under the SLL umbrella, and all of SLL projects are developed on Linux OS, this choice of OS for this project is limited to Linux. The Linux version used in this project is ubuntu-10.04-server.

The reason for choosing the server edition instead of the desktop edition is simply because during the installation of the former, the DNS server and LAMP server (Linux, Apache, MySQL and PHP) are also installed and only need to be configured to suit the project's requirements, unlike installing the desktop edition which will require installing the servers afterwards.

9.2. Modem

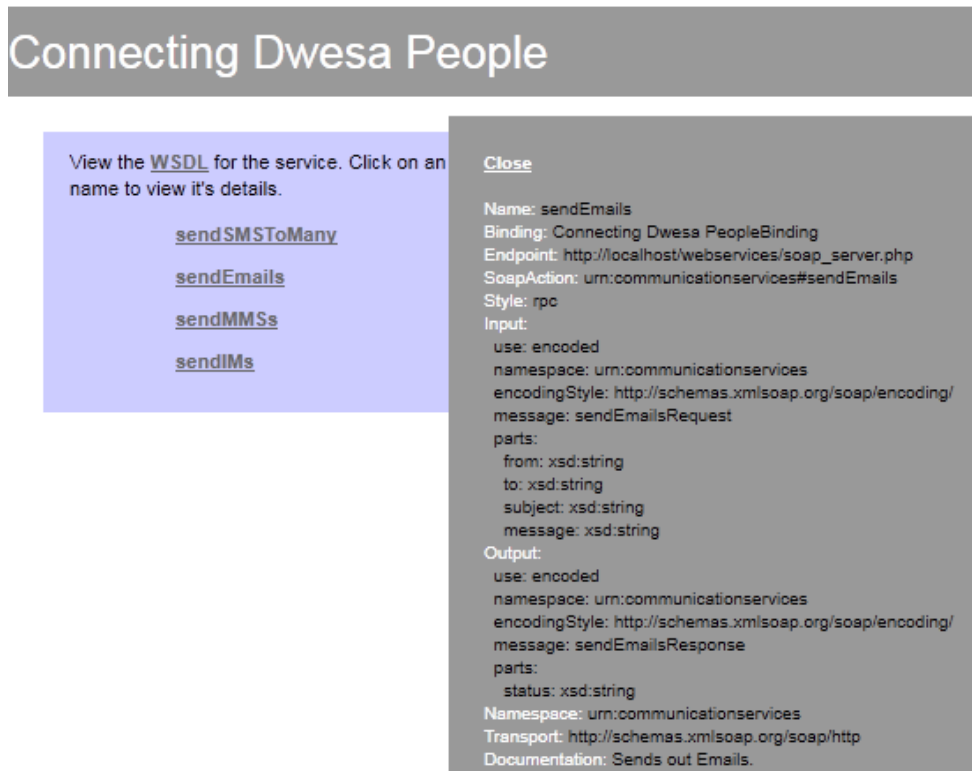
The modem used in this project is the Wavecom Fastrack Supreme 10 modem. The package comprises of an antenna, modem, power adapter, and a RS-232 cable as seen in the images below.



10. APPENDIX B - System Implementation Screenshots

10.1. NuSOAP interface

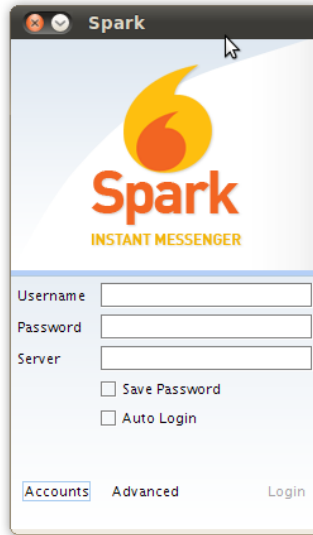
The image below shows the NuSOAP interface displaying the Email Web Service description.



10.2. The IM Client

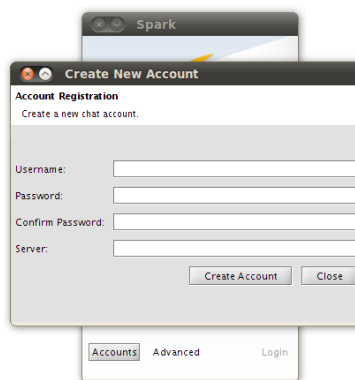
The IM system on this project uses spark as its client interface to connect to the system's XMPP server.

The images below show the IM client UI and the UI used to create account.



The image above is the Spark interface used to log into a XMPP server and it has a link, named accounts, that direct the user to the 'create account' interface. The name of the server needs to be specified in the server field so that Spark knows which XMPP server to connect to. In the case of this project, the server name is localhost.

The image below is the Spark interface used to create an account. Even in the case of creating an account, the server's name must be specified. This is simply because the user's details are saved in the XMPP server and not in Spark.



11. APPENDIX C - Configuration files

11.1. Kannel.conf

The following is Kannel's configuration file used in this project.

```
group = core
admin-port = 13000
admin-password = bar
status-password = foo
smsbox-port = 13001
wapbox-port = 13002
log-file = "/tmp/kannel.log"
log-level = 0
wdp-interface-name = "*"
store-location = "/var/log/kannel/kannel.store"

group = wapbox
bearerbox-host = localhost
log-file = "/tmp/wapbox.log"
syslog-level = none
access-log = "/tmp/wapaccess.log"
timer-freq = 10
map-url = "http://mmsc/* http://localhost:1981/*"

group = smsc
smsc = at
modemtype = auto
device = /dev/ttyS0
validityperiod = 167

group = smsbox
bearerbox-host = localhost
smsbox-id = box
sendsms-port = 13013
sendsms-chars = "0123456789 +-"
```

```

global-sender = 13013
log-file = "/var/log/kannel/smsbox.log"
log-level = 0
access-log = "/var/log/kannel/access.log"
sendsms-url = /cgi-bin/sendsms

group = sendsms-user
username = jimmy
password = junior
user-allow-ip = "*. *.*.*"
max-messages = 3
concatenation = true

group = sms-service
keyword-regex = .*
catch-all = yes
max-messages = 3
get-url = "http://localhost/sms?phone=%p&text=%a"

include = "/etc/kannel/modems.conf"

```

The *modems.conf* configuration is part of the Kannel's package and it does not need to be edited. The only requirement is to move it to the same directory as the main configuration file.

11.2. *modems.conf*

```

# Modems configuration
# Example and default values

group = modems
id = generic
name = "Generic Modem"

group = modems

```

```
id = wavecom  
name = Wavecom  
detect-string = "WAVECOM"  
  
group = modems  
id = premicell  
name = Premicell  
detect-string = "PREMICEL"  
no-pin = true  
no-smsc = true  
  
group = modems  
id = siemens_tc35  
name = "Siemens TC35"  
detect-string = "SIEMENS"  
detect-string2 = "TC35"  
init-string = "AT+CNMI=1,2,0,1,1"  
speed = 19200  
enable-hwvs = "AT\Q3"  
need-sleep = true  
  
group = modems  
id = siemens_m20  
name = "Siemens M20"  
detect-string = "SIEMENS"  
detect-string2 = "M20"  
speed = 19200  
enable-hwvs = "AT\Q3"  
keepalive-cmd = "AT+CBC;+CSQ"  
need-sleep = true  
  
group = modems  
id = siemens_sl45  
name = "Siemens SL45"
```



```
detect-string = "SIEMENS"
detect-string2 = "SL45"
init-string = "AT+CNMI=1,2,2,2,1"
keepalive-cmd = "AT+CBC;+CSQ"
speed = 19200
enable-hwvs = "AT\\Q3"
need-sleep = true
message-storage = "SM"

group = modems
id = nokiaphone
name = "Nokia Phone"
detect-string = "Nokia Mobile Phone"
need-sleep = true
keepalive-cmd = "AT+CBC;+CSQ"
enable-mms = true

group = modems
id = falcom
name = "Falcom"
detect-string = "Falcom"

group = modems
id = ericsson_r520m
name = "Ericsson R520m"
detect-string = "R520m"
init-string = "AT+CNMI=3,2,0,0"

group = modems
id = ericsson_t68
name = "Ericsson T68"
detect-string = "T68"
init-string = "AT+CNMI=3,3"
keepalive-cmd = "AT+CBC;+CSQ"
```

broken = true

group = modems

id = sonyericsson_gr47

name = "Sony Ericsson GR47"

detect-string = "GR47"

message-storage = "ME"

init-string = "AT+CNMI=3,2,0,0"

reset-string = "ATZ"

broken = true

group = modems

id = alcatel

name = "Alcatel"

detect-string = "Alcatel"

init-string = "AT+CNMI=3,2,0,0"

group = modems

id = sonyericsson_T630-T628

name = "Sony Ericsson T630-T628?"

init-string = "AT+CNMI=2,3,2,0,0;+CMGF=0?"

keepalive-cmd = "AT+CBC;+CSQ;+CMGF=0?"

broken = true

group = modems

id = sonyericsson_pli

name = "Sony Ericsson Pli"

detect-string = "Sony Ericsson Pli"

init-string = "ATQ0 V1 E1 S0=0 &C1 &D2 +FCLASS=0"

message-storage = "SM"

reset-string = "ATZ"

broken = true

11.3. *mmsbox.conf*

The following is the configuration file used to set Mbuni as a VAS gateway.

```
group = core
log-file = /var/log/mmsbox.log
access-log = /var/log/mmsbox-access.log
log-level = 0

group = mbuni
storage-directory = /var/spool/mbuni
max-send-threads = 5
maximum-send-attempts = 50
default-message-expiry = 360000
queue-run-interval = 5
send-attempt-back-off = 300
sendmms-port = 10001

group = mmsc
id = local
mmsc-url = http://mbuni:test@localhost:1982/soap
incoming-username = jimmy
incoming-password = junior
incoming-port = 12345
type = soap

group = mms-service
name = me
post-url = http://localhost/~bagyenda/test-mbuni.php
catch-all = true
http-post-parameters = fx=true&images[]=%i&text[]=%t&skip=1
accept-x-mbuni-headers = true
pass-thro-headers = X-NOKIA-MMSC-Charging,X-NOKIA-MMSC-Charged-Party
keyword = test
omit-empty = no
```

```
suppress-reply = true
service-code = regular

group = mms-service
name = fullmessage
get-url = http://localhost/images/apache_pb.gif
accept-x-headers = true
keyword = thixs

group = send-mms-user
username = jimmy
password = junior
faked-sender = 100
```

11.4. DNS configuration files

This section shows all the DNS configuration files except for the one displayed in section 5.5.2

- Defining the forward and reverse zones in */etc/bind/named.conf.local*

```
zone "example.com" {
    type master;
    file "/etc/bind/db.dwesaproject.com";
};
zone "56.20.172.in-addr.arpa" {
    type master;
    file "/etc/bind/db.172";
};
```

- The reverse zone file named *db.172*

\$TTL 604800

@ IN SOA ns.dwesaproject.com. root.dwesaproject.com. (
2010112900 ; Serial
604800 ; Refresh
86400 ; Retry
2419200 ; Expire
604800) ; Negative Cache TTL

;

@ IN NS ns.

68 IN PTR ns.dwesaproject.com.

12. APPENDIX D – Code Snippets

This section shows some of the code snippets that control how the system runs. Some of the codes snippets are displayed using more than one image because of the length of the code.

12.1. SOAP-server

The images below show how the SOAP-server is implemented in this project.

```
1 <?php
2 require_once("nusoap.php");
3 require_once("functions.php");
4 $debug = 1;
5 $server = new soap_server();
6 $server->configureWSDL('Connecting Dwesa People',
7 'urn:communicationservices');
8 $server->register(
9     'sendSMSToMany',
10    array('to' => 'xsd:string', 'from' => 'xsd:string',
11          'message' => 'xsd:string'),
12    array('status' => 'xsd:string'),
13    'urn:communicationservices',
14    'urn:communicationservices#sendSMSToMany',
15    'rpc',
16    'encoded',
17    'Sends the same SMS to multiple phone numbers.
18    Give your 10 digit phone number for user ID.
19    Separate each phone number with a semicolon(\';\').'
20    );
21
22 $server->register(
23     'sendEmails',
24    array('from' => 'xsd:string', 'to' => 'xsd:string',
25          'subject' => 'xsd:string', 'message' => 'xsd:string'),
26    array('status' => 'xsd:string'),
27    'urn:communicationservices',
28    'urn:communicationservices#sendEmails',
29    'rpc',
30    'encoded',
31    'Sends out Emails.'
32    );
33
34 $server->register(
35     'sendMMSs',
```

```

36 array('imageBytes' => 'xsd:base64Binary', 'imageName' =>
37 'xsd:string', 'soundBytes' => 'xsd:base64Binary',
38 'soundName' => 'xsd:string', 'from' => 'xsd:string', 'to' =>
39 'xsd:string', 'subject' => 'xsd:string', 'message' => 'xsd:string'),
40 array('status' => 'xsd:string'),
41 'urn:communicationservices',
42 'urn:communicationservices#sendMMSs',
43 'rpc',
44 'encoded',
45 'Sends out MMSs.'
46 );
47
48 $server->register(
49 'sendIMs',
50 array('to' => 'xsd:string', 'message' => 'xsd:string'),
51 array('status' => 'xsd:string'),
52 'urn:communicationservices',
53 'urn:communicationservices#sendIMs',
54 'rpc',
55 'encoded',
56 'Sends out IM notifications.'
57 );
58
59 function sendSMSToMany($from, $to, $msg){
60     return new soapval('return', 'xsd:string', sendSMS($from, $to, $msg));
61 }
62
63 function sendEmails($from, $to, $subject, $message){
64     return new soapval('return', 'xsd:string', sendEmail($from, $to, $subject,
65 $message));
66 }
67
68 function sendMMSs($file1, $name1, $file2, $name2, $from, $to, $subject,
69 $message){
70     return new soapval('return', 'xsd:string', sendMMS($file1, $name1, $file2,
71 $name2, $from, $to, $subject, $message));
72 }
73
74 function sendIMs($to, $message){
75     return new soapval('return', 'xsd:string', sendIM($to, $message));
76 }
77
78 $HTTP_RAW_POST_DATA = isset($HTTP_RAW_POST_DATA) ? $HTTP_RAW_POST_DATA : '';
79 $server->service($HTTP_RAW_POST_DATA);
80 ?>

```

12.2. Web Services functions

This section shows all the Web Services functions that execute requests sent by the SOAP-client and return responses.

- sendSMS

```
3 function sendSMS($from, $to, $msg) {
4     $gw_host = '127.0.0.1';
5     $gw_user = 'jimmy';
6     $gw_pass = 'junior';
7     $gw_port = '13013';
8     $timeout = 30;
9     $ret = "";
10
11     $msg = urlencode($msg);
12     $from = urlencode($from);
13     $p = urlencode($to);
14     $result = curl_init("http://$gw_host:$gw_port/cgi-bin/sendsms?
15     username=$gw_user&password=$gw_pass&from=$from&to=$p&text=$msg&
16     coding=2&charset=iso-8859-7");
17     if (curl_exec($result)) {
18         return "<center><font size=\"2\" color=\"blue\">SMS sent.
19         </font></center>";
20     }else{
21         return "<center><font size=\"2\" color=\"red\">SMS not sent.
22         Try again.</font></center>";
23     }
24 }
```


- sendMMS

```
46 function sendMMS($file1, $name1, $file2, $name2, $from, $to, $subject, $message){
47     $fp = fopen("image/$name1", "w");
48     fwrite($fp, base64_decode($file1));
49     fclose($fp);
50     $fp = fopen("sound/$name2", "w");
51     fwrite($fp, base64_decode($file2));
52     fclose($fp);
53     $myfile = "message.txt";
54     chmod("$myfile", 0777);
55     $fh = fopen($myfile, 'w') or die ($err = "Can't open file");
56     fwrite($fh, $message);
57     fclose($fh);
58     $from = urlencode($from);
59     $pharr = explode(";", $to);
60     foreach ($pharr as $p){
61     if (strlen($p) != 10 || !is_numeric($p) || strpos($p, ".") != false){
62         $ret .= "invalid number: " . $p . "\n";
63         continue;
64     }
65     $p = urlencode($p);
66     $result = curl_init("http://$gw_host:$gw_port/cgi-bin/sendmms?username=
67     $gw_user&password=$gw_pass&from=$from&to=$p&content=$msg&coding=2
68     &charset=iso-8859-7");
69     if (curl_exec($result)){
70         $ret .= "Done" . $ret;
71         return $ret;
72     }else{
73         $ret .= "SMS not sent. Error occured";
74         return $ret;
75     }
76     }
77 }
```

- sendEmail

```
43 function sendEmail ($from, $to, $subject, $message, $bytes, $filename){
44     $bytes = base64_decode($bytes);
45     $fp = fopen("/var/www/webservices/uploads/$filename", "w");
46     fwrite($fp, base64_decode($bytes));
47     fclose($fp);
48     $fileatt = "/var/www/webservices/uploads/$filename";
49     $fileatt_type = "application/octet-stream";
50     $fileatt_name = $filename;
51     $email_from = $from;
52     $email_subject = $subject;
53     $email_txt = $message;
54     $email_to = $to;
55     $headers = "From: ".$email_from;
56     $file = fopen($fileatt, 'rb');
57     $data = fread($file, filesize($fileatt));
58     fclose($file);
59     $semi_rand = md5(time());
60     $mime_boundary = "==Multipart_Boundary_x{$semi_rand}x";
61     $email_message .= "{$message}\n\n" .
62     $email_message . "\n\n";
63     $data = chunk_split(base64_encode($data));
64     $email_message .= "--{$mime_boundary}\n" .
65     "Content-Type: {$fileatt_type};\n" .
66     " name=\"{$fileatt_name}\";\n" .
67     "Content-Transfer-Encoding: base64\n\n" .
68     $data . "\n\n" .
69     "--{$mime_boundary}--\n";
70     $ok = @mail($email_to, $email_subject, $email_message, $headers);
71     if($ok) {
72         return "The file was successfully sent.</font></center>";
73     } else {
74         return "Your message was not sent.</font></center>";
75     }
76 }
```

12.3. IM Web Service Classes

This section shows the different classes used in the implementation of the IM Web Service.

- Spark.php

The *spark.php* file includes a class named Spark which is responsible for handling communications between the system and the Openfire XMPP server. It is used to send IM notifications to users on Spark.

```
1 <?php
2 require_once("XMPPHP/XMPP.php");
3 class Spark {
4     private $conn;
5     function __construct() {
6         $this->conn = new XMPPHP_XMPP('samalenge', 5222,
7             'admin', 'junior', 'xmpphp', 'pingstream',
8             $printLog=false, $loglevel=0);
9     }
10    function connect() {
11        try {
12            $this->conn->connect();
13            $this->conn->processUntil('session_start');
14        } catch(XMPPHP_Exception $e) {
15            die($e->getMessage());
16        }
17    function disconnect() {
18        try {
19            $this->conn->disconnect();
20        } catch(XMPPHP_Exception $e) {
21            die($e->getMessage());
22        }
23    function send_message($to, $msg) {
24        $this->connect();
25        try {
26            $this->conn->message($to, $msg);
27        } catch(XMPPHP_Exception $e) {
28            die($e->getMessage());
29        }
30        $this->disconnect();
31    }
32 }
33 ?>
```

- gtalk.php

The *gtalk.php* file includes a class named GTalk which is responsible for handling the communication between the system and Google Talk (Google XMPP server).

```
1 <?php
2 require_once("XMPPHP/XMPP.php");
3 class GTalk {
4     private $conn;
5     function __construct() {
6         $this->conn = new XMPPHP_XMPP('talk.google.com',
7             5222, 'USERNAME', 'PASSWORD', 'xmpphp', 'gmail.com',
8             $printLog=false, $loglevel=0);
9     }
10    function connect() {
11        try {
12            $this->conn->connect();
13            $this->conn->processUntil('session_start');
14        } catch(XMPPHP_Exception $e) {
15            die($e->getMessage());
16        }
17    function disconnect() {
18        try {
19            $this->conn->disconnect();
20        } catch(XMPPHP_Exception $e) {
21            die($e->getMessage());
22        }
23    function send_message($to, $msg) {
24        $this->connect();
25        try {
26            $this->conn->message($to, $msg);
27        } catch(XMPPHP_Exception $e) {
28            die($e->getMessage());
29        }
30        $this->disconnect();
31    }
32 }
33 ?>
```

12.4. The SOAP-clients

This section shows all the SOAP-clients implemented in this project except the sms.php which is already discussed in the implementation chapter (Chapter 5).

- email.php

```
7 <?php
8 if (isset($_POST['submitEmail'])) {
9     $errEmail = "";
10    $errMsg = "";
11    $err = "";
12    $from = $_POST["from"];
13    $to = $_POST["to"];
14    $subject = $_POST["subject"];
15    $message = $_POST["message"];
16
17    if (preg_match('/^(?:(\w\!#\$\%\&\'\*\+\-\\/\=\?\^\`\{\|\}\~]+\.)+
18    [\w\!#\$\%\&\'\*\+\-\\/\=\?\^\`\{\|\}\~]+@(?:(?:(?:[a-zA-Z0-9_
19    (?:[a-zA-Z0-9_\-](?!\.))){0,61}[a-zA-Z0-9_]?\.)+[a-zA-Z0-9_
20    (?:[a-zA-Z0-9_\-](?!$)){0,61}[a-zA-Z0-9_]?|{
21    ?:\[(?:(?:[01]?[d]{1,2}|2[0-4][d]{1}|25[0-5])\.)\{3\}(?:[01]?[d]{1,2}
22    |2[0-4][d]{1}|25[0-5])\})\]$/',$toEm))=== 0)
23        $errEmail = "<font size=\"-2\" color=\"red\">
24        Email must comply with this mask: chars(.chars)@chars(.chars)
25        .chars(2-4)</font><br />";
26
27    if (!empty($errEmail)){
28        include ('includes/emailFrm.php');
29    }else{
30
31        //make sure that 'nusoap/nusoap.php' is in the include path
32        require_once('nusoap.php');
33
34        //require_once('nusoapmime.php');
35    if (!empty($_FILES['uploadedfile']['name'])){
36        $bytes = base64_encode($_FILES['uploadedfile']['name']);
37        $filename = basename($_FILES['uploadedfile']['name']);
38        $client = new nusoap_client('http://localhost/webservices/
39        soap_server.php?wsdl', true);
40        $err = $client->getError();
```

```

41
42     if ($err) {
43         echo 'Creation of the wrapper failed<pre>' . $err .
44             '</pre>';
45     }
46
47     $result = $client->call(
48         'sendEmailsWithAttachment',
49         array('from' => $from, 'to' => $to,
50             'subject' => $subject, 'message' => $message,
51             'bytes' =>$bytes , 'filename' => $filename),
52             'urn:communicationservices',
53             'urn:communicationservices#SendEmailsWithAttachment'
54         );
55     if ($client->fault) {
56         echo '<h2>Fault (Expect - The request contains an invalid SOAP
57 body)</h2><pre>'; print_r($result); echo '</pre>';
58     } else {
59         $err = $client->getError();
60         if ($err) {
61             $status = $err;
62             include ('includes/emailFrm.php');
63         } else {
64             $status = $result;
65             include ('includes/emailFrm.php');
66         }
67     }
68
69 }else{
70     $client = new nusoap_client('http://localhost/webservices/
71 soap_server.php?wsdl', true);
72     $err = $client->getError();
73
74     if ($err) {

```

```

75     echo 'Creation of the wrapper failed<pre>' . $err .
76     '</pre>';
77 }
78
79 $result = $client->call(
80     'sendEmails',
81     array('from' => $from, 'to' => $to,
82         'subject' => $subject, 'message' => $message),
83         'urn:communicationservices',
84         'urn:communicationservices#SendEmails'
85     );
86 if ($client->fault) {
87     echo '<h2>Fault (Expect - The request contains an invalid
88 SOAP body)</h2><pre>'; print_r($result); echo '</pre>';
89 } else {
90     $err = $client->getError();
91     if ($err) {
92         $status = $err;
93         include ('includes/emailFrm.php');
94     } else {
95         $status = $result;
96         include ('includes/emailFrm.php');
97     }
98 }
99 }
100 }
101 }else{
102     include ('includes/emailFrm.php');
103 }
104 ?>

```

- im.php

```

7  <?php
8  if (isset($_POST['submitIM'])) {
9      //make sure that 'nusoap/nusoap.php' is in the include path
10
11     require_once('nusoap.php');
12     $to = $_POST["to"];
13     $message = $_POST["smsmessage"];
14     $status = "";
15     $errTo = "";
16     $errMsg = "";
17
18     if (trim($to) == "" || strlen($to) == 0){
19         $errTo .= "<font size=\"-2\" color=\"red\">Please enter recipient's address.</font>";
20     }
21
22     if (trim($message) == "" || strlen($message) == 0){
23         $errMsg .= "<font size=\"-2\" color=\"red\">Invalid message.</font>";
24     }
25
26     if (!empty($errTo) || !empty($errMsg)){
27         include ('includes/imFrm.php');
28
29     }else{
30         $client = new nusoap_client('http://localhost/webservices/soap_server.php?wsdl', true);
31         $err = $client->getError();
32         if ($err) {
33             echo '<h2>Constructor error</h2><pre>' . $err . '</pre>';
34             echo '<h2>Debug</h2><pre>' . htmlspecialchars($client->getDebug(), ENT_QUOTES) . '</pre>';
35             exit();
36         }
37         $result = $client->call(
38             'sendSMStoMany',
39             array('to' => $to, 'message' => $message),
40             'urn:communicationservices',
41             'urn:communicationservices#sendSMStoMany'
42         );
43
44         if ($client->fault) {
45             echo '<h2>Fault (Expect - The request contains an invalid SOAP body)</h2><pre>';
46             print_r($result);
47             echo '</pre>';
48         } else {
49             $err = $client->getError();
50             if ($err) {
51                 $status = "<font size=\"1\" color=\"red\"> { $err } </font>";
52                 include ('includes/imFrm.php');
53             } else {
54                 $status = ptint_r($result);
55                 include ('includes/imFrm.php');
56             }
57         }
58     }
59 }
60
61 }else{
62 include ('includes/imFrm.php');
63 }
64
65 ?>

```


13. APPENDIX E – Further system testing and results

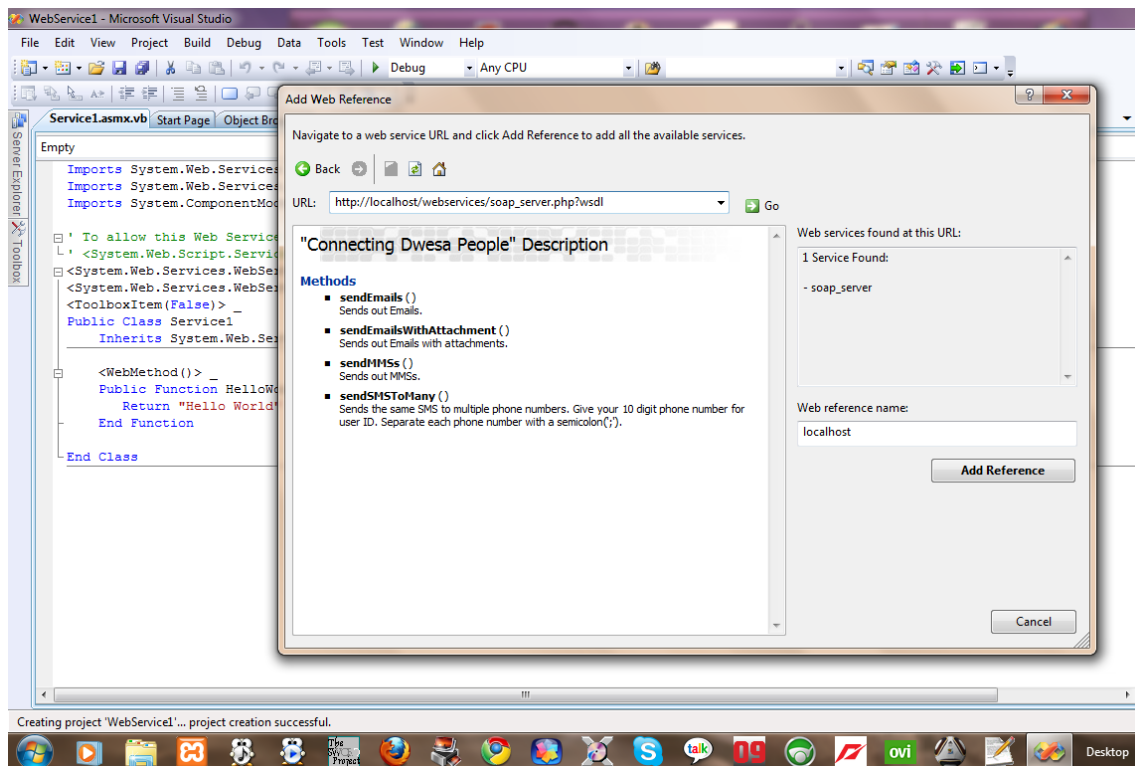
13.1. Backend process testing

The image below shows the testing of the Postfix configuration file. The test is done by restarting the Postfix process, and if it restarts successfully this means that the Postfix is properly configured.

```
root@jimmysamalenge:/home/jimmy# cd ~
root@jimmysamalenge:~# /etc/init.d/postfix restart
* Stopping Postfix Mail Transport Agent postfix [ OK ]
* Starting Postfix Mail Transport Agent postfix [ OK ]
root@jimmysamalenge:~#
```

13.2. Compatibility testing

The image below shows a Visual Basic application which connects to this system SOAP-server and requests to make use of its web Services.

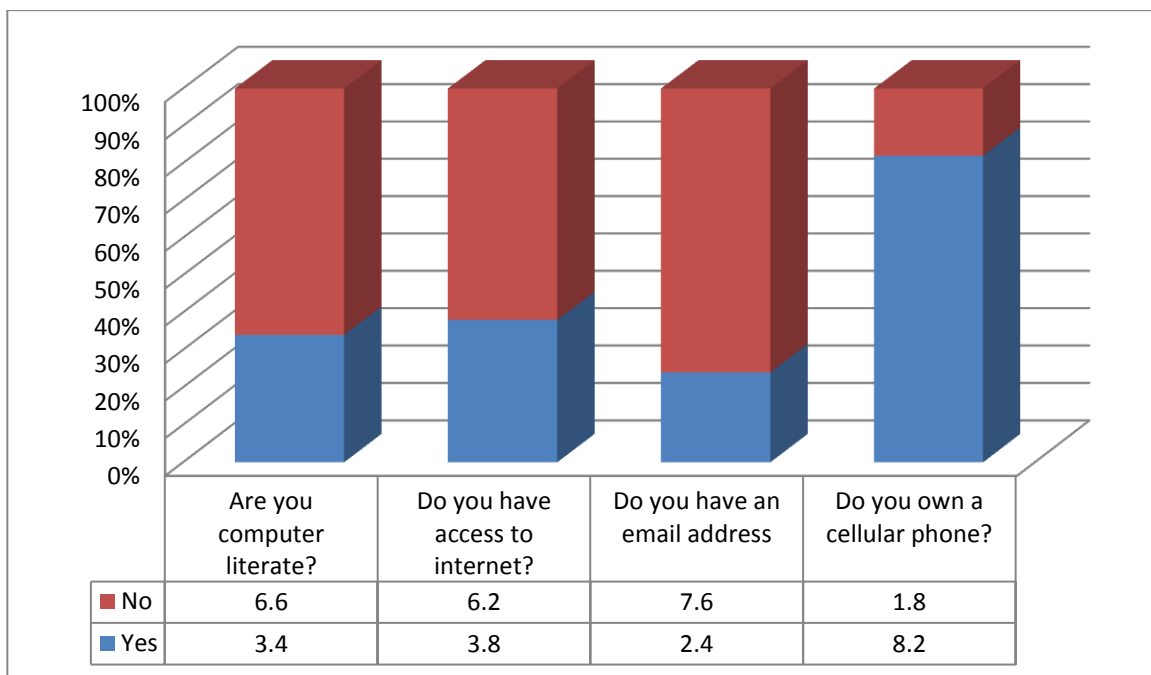


14. APPENDIX F – Further usability testing results

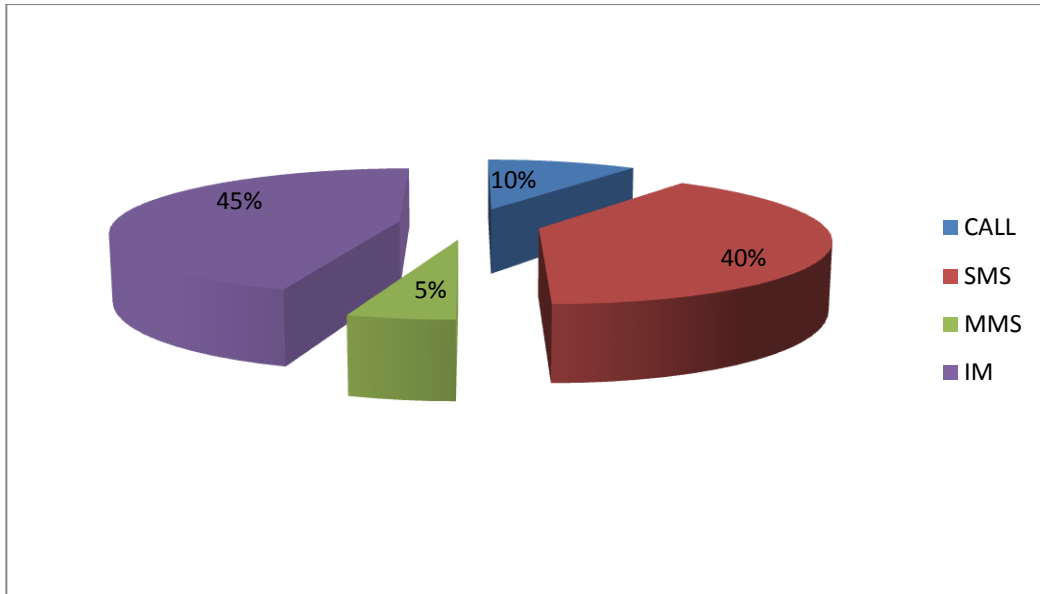
14.1. System Usability Scale: the questionnaire

14.1.1. Part one: User's background

Question	Yes	No
Are you computer literate?	34%	66%
Do you have access to the Internet?	38%	62%
Do you have an email address?	24%	76%
Do you own a cell phone?	82%	18%



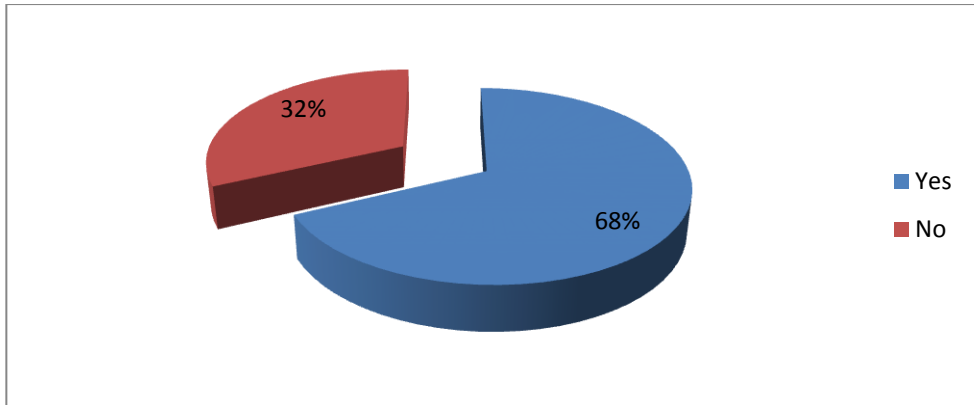
Which mobile services do you use the most?							
CALL	10%	SMS	40%	MMS	5%	IM	45%



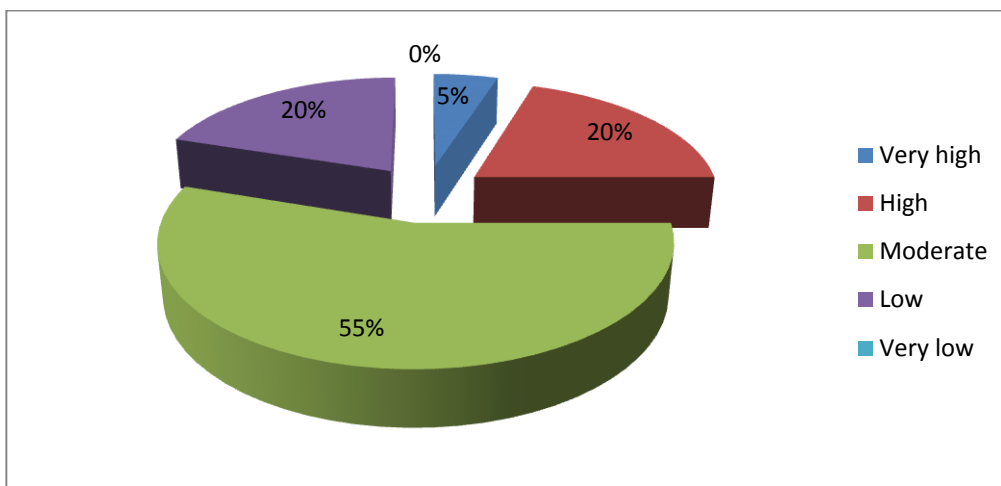
14.1.2. Part two: SUS

1. Do you think that this application easy to use?				Yes	No
				32%	68%
2. How much effort did you personally put forth to use the application?	Very high	High	Moderate	Low	Very Low
	5%	20%	55%	20%	0%
3. How easy is it to understand the user interface?	Very difficult	Difficult	Moderate	Easy	Very Easy
	0%	2%	64%	24%	0%
4. How do you rate the entire system?	Needs improvement	Adequate	Good	Very good	Excellent
	0%	4%	56%	40%	0%

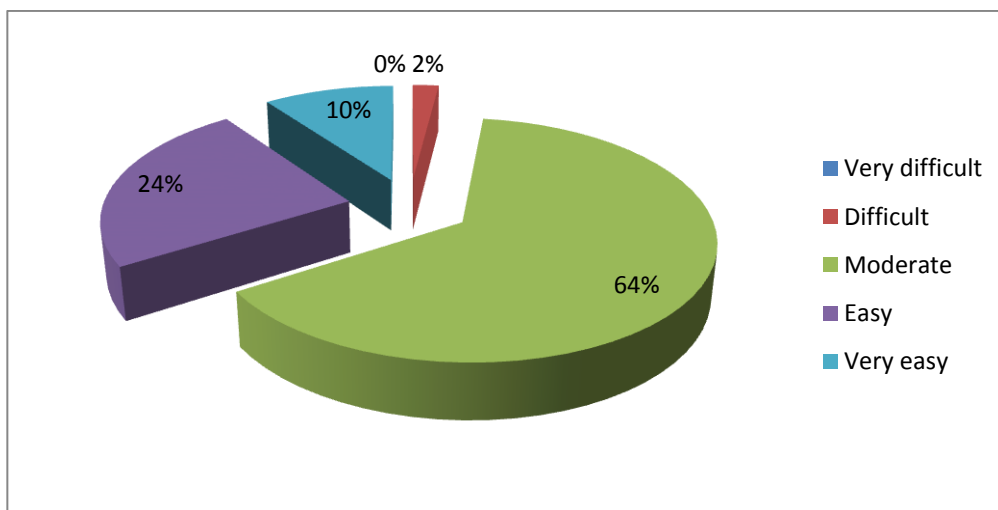
- Question 1: Do you think that this application easy to use?



- Question 2: How much effort did you personally put forth to use the application?



- Question 3: How easy is it to understand the user interface?



- Question 4: How do you rate the entire system?

