REMOTE MONITORING AND MANAGEMENT OF SERVERS FROM MOBILE DEVICES

BY

AHMAD, BARROON ISMA'EEL

DEPARTMENT OF MATHEMATICS

AHMADU BELLO UNIVERSITY, ZARIA

NIGERIA

FEBRUARY, 2011

REMOTE MONITORING AND MANAGEMENT OF SERVERS FROM MOBILE DEVICES

BY

AHMAD, BARROON ISMA'EEL

M.Sc./SCIEN/00719/2008-2009

A THESIS SUBMITTED TO THE

POSTGRADUATE SCHOOL,

AHMADU BELLO UNIVERSITY, ZARIA

NIGERIA

IN PARTIAL FULFILMENT FOR THE AWARD

OF MASTER OF SCIENCE IN

COMPUTER SCIENCE

DEPARTMENT OF MATHEMATICS

AHMADU BELLO UNIVERSITY, ZARIA

NIGERIA

JUNE, 2011

DECLARATION

I declare that the work in the thesis entitled ‘Remote Monitoring and Management of Servers from Mobile Devices’ has been performed by me in the Department of Mathematics under the supervision of Dr. M. I. Seyed Buhari and Prof. S. B. Junaidu.

The information derived from the literature has been duly acknowledged in the text and a list of references provided. No part of this thesis was previously presented for another degree or diploma to the best of my knowledge at any University.

Ahmad Barroon Isma’eel	_____	_____
Name of Student	Signature	Date

CERTIFICATION

This thesis entitled “REMOTE MONITORING AND MANAGEMENT OF SERVERS FROM MOBILE DEVICES” by Ahmad Barroon Isma’eel meets the regulations governing the award of the degree of Master of Science of Ahmadu Bello University, and is approved for its contribution to knowledge and literary presentation.

Dr. M. I. Seyed Buhari _____
Major Supervisor Date

Prof. S. B. Junaidu _____
Minor Supervisor Date

_____
External Examiner Date

Dr. A. A. Tijjani _____
Head of Department Date

Prof. A. A. Joshua _____
Dean of Postgraduate School Date

DEDICATION

This work is dedicated to my dearest Parents, Brothers and Sisters.

ACKNOWLEDGEMENT

Thanks, glorification and total submission are due to ALLAH (SWT) who has made it possible for me to finish this work.

My sincere gratitude goes to my great supervisors and mentors Dr. M. I. Seyed Buhari for a wonderful guide and being patient with me during this research, Prof. S. B. Junaidu for his encouragement and fatherly advice.

I would like to extend my thanks to the entire staff members of Mathematics Department, My lecturers and colleagues for their useful suggestions. This thesis has been completed with support from the ABU MacArthur Foundation Projects.

My deepest appreciation goes to my adorable parents, brothers (Dr. Jameel, Engr. Mashhood, Awwal …), sisters (Nasiba, Aisha, Ummi, Jameela, Ummi…), cousins (Zainab…) and my entire family members. Thanks for your support and prayers, may Allah reward you with best of reward (Aljannah).

I am indebted to the likes of Dr. Bashir U. Kurfi, Dr. Aminu A. Mohd, M. Shuaibu Umar, Mr. Syazwan Hisham Mohd Ridzwan, Engr. Yusuf and M. Aliyu (EMS) whose help directly or indirectly contributed to the success of this work, thank you all.

Finally I would like to thank my friends like Bagiwa, Ibrahim, Abdullahi, Murtala, Dishing, Umar and Shehu for their advice and prayers.

ABSTRACT

Internet usage has tremendously increased these days. The services that users access across a network and the servers that host these services need to be monitored for availability/protection and efficiency. The need for remote monitoring of servers as well as the services is essential, since this will notify the user/administrator instantly when something happens to the server, also the user can be anywhere to monitor the server. The development of a remote server monitoring system that permits an administrator to check the status of a server from a mobile phone is the subject of this research. The application was designed using Java technology (Java 2 Micro Edition). Access to the server is provided through an SMS gateway, which is connected to the network through the wired network. The application also enables the administrator to restart a machine, after power outage, which is common in our locality. The system that achieved the aforementioned objectives was developed and tested with the attendant benefits of reducing the time and overhead of server management.

TABLE OF CONTENTS

DECLARATION ii
CERTIFICATION iii
DEDICATION iv
ACKNOWLEDGEMENT v
ABSTRACT vi
TABLE OF CONTENTS vii
LIST OF TABLES ix
LIST OF FIGURES x
LIST OF APPENDICES xii
ABBREVIATION xiii
CHAPTER ONE: GENERAL INTRODUCTION 1
1.1 Background to Study 1
1.2 Problem Definition 2
1.3 Objective of the Thesis 2
1.4 Methodology 3
1.5 Research Questions 3
1.6 Scope 3
1.7 Contribution to Knowledge 4
1.8 Organization of the Thesis 4
CHAPTER TWO: LITERATURE REVIEW 5
2.1 Overview of Server Management 5
2.2 Messaging Gateway 5
2.3 SMS 10
2.4 J2ME 11
2.5 WMIC 15
2.6 Wake On LAN 18
2.7 Related Work 19
2.8 Conclusion 26
CHAPTER THREE: DESIGN AND IMPLEMENTATION 27

3.1 System Architecture 27
3.2 Implementation 28
3.2.1 Frontend Architecture 28
3.2.2 Backend Architecture 36
3.2.2.1 Setting up Gammu SMS Gateway 37
3.2.2.2 SMS Handling 45
3.2.2.3 Remote Command Execution 48
3.2.2.4 Power Management 49
3.2.2.5 Tools Used 53
3.3 Conclusion 53
CHAPTER FOUR: DICUSSION AND EVALUATION 55
4.1 Development Environment 55
4.1.1 Hardware Specification 55
4.1.2 Software Specification 55
4.2 Functional Structure of the System 56
4.3 Performance Evaluation 74
4.4 Conclusion 74
CHAPTER FIVE: SUMMARY, CONCLUSION AND RECOMMENDATION 75
5.1 Summary 75
5.2 Future Work 76
5.3 Conclusion 76
REFERENCES 78
APPENDIX A: SETTING UP GAMMU ON WINDOWS MACHINE 83
APPENDIX B: CREATING AND RUNNING A PROJECT IN WTK 84
APPENDIX C: SOURCE CODE 86

LIST OF TABLES

Table 3.1: JAD and Manifest files attributes 35
Table 3.2: Daemon table structure 42
Table 3.3: Gammu table structure 42
Table 3.4: Inbox table structure 42
Table 3.5: Outbox table structure 43
Table 3.6: Outbox_Multipart table structure 44
Table 3.7: Phone table structure 45
Table 3.8: Sentitem table structure 45
Table 3.9: Parallel port pins and their function 51
Table 4.1: Commands required to be typed by the user 64

LIST OF FIGURES

Figure 2.1a: Handling SMS that support same protocol 7
Figure 2.1b: Handling SMS that support different protocol 7
Figure 2.2: Platforms and targeted market. 13
Figure 2.3: J2ME configuration, profiles and optional packages 16
Figure 2.4: WMIC specifying command in interactive mode. 18
Figure 2.5: WMIC specifying command in batch mode 19
Figure 3.1: General system architecture protocol 29
Figure 3.2: MIDlet life cycle.....31
Figure 3.3a: Opening a project.....33
Figure 3.3b: List of created projects including sample from WTK developers. 33
Figure 3.3c: Building (compiling) a project.....34
Figure 3.4: Preverification process.....34
Figure 3.5: Parallel port pins.....51
Figure 3.6a: Power management circuit.....53
Figure 3.6b: Complete circuit snapshot.....53
Figure 4.1: System process flow.....58
Figure 4.2: Confirmation message during installation process packages 59
Figure 4.2: Login interface.....60
Figure 4.3: Success message when supplying correct login details.....61
Figure 4.4: Error message when supplying invalid username or password.....62
Figure 4.5: Sample command typed by the user.....63
Figure 4.6a: Battery status process flow 66
Figure 4.6b: UPS status process flow.....66
Figure 4.7: Server status process flow.....67
Figure 4.8: Flow diagram for checking number of users. 68

Figure 4.9: Flow diagram for checking the users on the server.....69
Figure 4.10: Web server status flowchart.....71
Figure 4.11a: Starting the server by sending magic packet 72
Figure 4.11b: Starting the server by sending data to a circuit. 72
Figure 4.12a: Stopping the server by running a command in the server.....73
Figure 4.12b: Stopping the server by sending data to a circuit.....74
Figure 4.13: Process flow when user sends anything other than the commands specified.....74

LIST OF APPENDICES

APPENDIX A: SETTING UP GAMMU ON WINDOWS.....84
APPENDIX B: CREATING AND RUNNING A PROJECT IN WTK.....85
APPENDIX C: SOURCE CODE.....87

ABBREVIATION

3D	Three Dimensional
3G	Third Generation
3GPP	Third Generation Partnership Project
AMD	Advanced Micro Devices
API	Application Programming Interface
ASP	Active Server Pages
BIOS	Basic Input/Output System
CCD	Charged-Couple Device
CDC	Connected Device Configuration
CDMA	Code Division Multiple Access
CIMD	Computer Interface to Message Distribution
CLDC	Connected Limited Device Configuration
COM	Component Object model
CP	Communication Processor
CPU	Central Processing Unit
CVM	Compact Virtual Machine
DB9	Data Bus 9 pins
DC	Direct Current
DOS	Disk Operating System
ECG	Electrocardiography
EMAIL	Electronic Mail
EMI	External Machine Interface
ETSI	European Telecommunication Standards Institute
FAX	Facsimile
FBUS	Fast Bus
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communication
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IMP	Information Module Protocol
IP	Internet Protocol
J2EE	Java 2 Enterprises Edition
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JAD	Java Application Descriptor
JAR	Java Archive

JDK	Java Development Kit
JMF	Java Media Framework
JSR	Java Specification Request
JVM	Java Virtual Machine
KVM	Kilobyte Virtual Machine
LAN	Local Area Network
MAC	Media Access Control
MIB	Management Information Base
MIDP	Mobile Information Device Profile
MMS	Multimedia Message Service
NIC	Network Interface Card
OEM	Original Equipment Manufacturer
PC	Personal Computer
PDA	Personal Digital Assistant
PHP	Personal Home Page
PIC	Programmable Interface Controller
PIN	Personal Identification Number
PLC	Programming Logic controller
POP3	Post Office Protocol version 3
RAS	Remote Access Server
RS232	Recommended Standard 232
SIM	Subscriber Identity Module
SMPP	Short Message Peer to Peer
SMS	Short Message Service
SMSC	Short Message Service Center
SMSD	Short Message Service Daemon
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSH	Secure Shell
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
UCP	Universal Computer Protocol
UDP	User Datagram Protocol
UPS	Uninterruptible Power Supply
URL	Uniform Resource Locator
USB	Universal Serial Bus
VASP	Value Added Service Providers

VB	Visual Basic
WAP	Wireless Application Protocol
WfM	Wireless for Management
WMI	Windows Management Instrumentation
WMIC	Windows Management Instrumentation Command-line
WOL	Wake On LAN
WQL	WMI Query Language
WTK	Wireless Tool Kit
XML	Extensible Markup Language

CHAPTER ONE: GENERAL INTRODUCTION

1.1 Background to Study

Monitoring and management of Network and/or System is crucial because of the fact that most of our services especially web services are expected to be available to all clients round the clock. A specialized computer called Server provides these services to clients over a network. Certainty Solutions (2000) describes a System monitoring, as a mechanism to test, track, and report on the availability and condition of the system as well as services on the system. Generally, servers are monitored by keeping an eye on them; that is, the administrator has to either be with the server or with a device (can be a PC or mobile device) that remotely monitors the server.

The rapid growth of mobile and wireless technology makes mobile devices such as mobile phones to become part of our lives, as a result of which we are indirectly replacing PCs in so many ways like surfing the Internet, sending e-mails, remote controlling home appliances and the like. The communication technology used by our present mobile phones includes: GPRS, WAP, Bluetooth, SMS and MMS, out of which SMS is the popularly used technology.

According to Hord (2005), Short Message Service (SMS) is a communications protocol which allows the exchange of short text messages between mobile devices. This is one of the vital services provided by mobile GSM network; SMS was invented in order to overcome a communication problem like out of coverage or user's mobile phone turned off. Since mobile phones are becoming more advanced, there is a need for developing applications for mobile phones to make them more useful.

1.2 Problem Definition

The availability and reliability of services like email, web etc; provided by a server can be known by constant monitoring of the server that hosts these services. Meanwhile, constant power supply is essential in order to maximize the uptime of the servers; failure of the server may result to an irretrievable loss of data.

Nowadays administrators do not need to sit in front of a machine to make sure the system works, but rather they can use their mobile phones from anywhere at any time in order to keep track of their systems and as well as services.

This can be achieved by designing an application that allows the administrator to monitor his server from a mobile phone, without the need for the use of a client PC. The server can only be accessed from a mobile that has the application installed; and besides, the user has to be authenticated before granting access to the application using a username and password that would be set in the application.

1.3 Objective of the Thesis

The main goal of this research is to develop a mobile application which will allow an administrator to monitor the status of the server (up/down), web service hosted on the server as well as the server's battery. The system will be developed such that, the server will be communicating with a machine configured as an SMS gateway, if the status of the server changes, automatically a notification in form of message will be sent to the administrator's mobile phone; and also if the administrator wishes to know the status at any point in time he can do that by sending a message to the machine. In essence, the communication pattern is between a server and a system (SMS gateway) and the gateway to a mobile phone or vice versa.

1.4 Methodology

The procedures adapted for the research work are listed below:

i. Java technology (J2ME) was used for developing the client application for sending SMS.
ii. On the remote machine: Gammu SMS Gateway, Batch files and WMIC were used.
iii. On the server: Shell script, awk and Wake On LAN were used.

1.5 Research Questions

We will answer the following questions in this thesis:

i. How do we monitor a server remotely?
ii. Can we setup an SMS gateway machine to communicate with both the server and the client?
iii. Is it possible to send a message from a mobile phone and perform an action on a computer based on the content of the message?
iv. How can we send a message automatically to administer services hosted on the server or to administer the server itself?

1.6 Scope

This research focused on developing remote monitoring and management system that will monitor the status of the server, services hosted on the server as well as the server's battery. We will use two computer systems in developing the application. The reasons for using two systems are:

i. Resource Minimization: One computer can be used to monitor multiple servers, for this reason only the computer that will monitor the servers will have the SMS gateway and other resources needed for the system.

ii. Eliminate Total System Failure: If one computer is used and unfortunately there is power outage, then the system will completely collapse. In that case, the notification cannot be sent to the administrator.

The first computer is the server that has the services required for the system, while the second computer serves as the monitoring system that will have SMS gateway (gammu) for manipulating the SMS connected with a mobile phone (nokia 3310) through serial cable.

1.7 Contribution to Knowledge

This research will be helpful to network administrators by making it possible to monitor the network, server and services like web service remotely on mobile phone. This makes life easier for administrators since they do not have to be physically around at their server's location to administer them. Furthermore, server up-time and services availability will be increased.

1.8 Organization of the Thesis

Chapter two looks into relevant literature related to the work, and also discusses the technology used in developing the system.

Chapter three discusses how to develop the server monitoring and management system to work with mobile phones.

Chapter four discusses the environment in which the system was implemented; it also focused on the functional structure of the whole system and the performance of the system.

Chapter five presents the summary, future work and conclusion for the thesis.

CHAPTER TWO: LITERATURE REVIEW

2.1 Overview of Server Management

With the increase in popularity and technological expansion of computer networks, the data that organizations, educational institutions and governments used in running their affairs have to be kept in a computer known as a Server. These data should be made available so that people can access them not necessarily on the machine that they are located, but rather from another office, at home or anywhere in the world. There is the need for these servers to be managed properly in order to have good decision making in an organization. Management of these servers involves making sure that the services on the server are running as expected, also things like disk utilization, bandwidth utilization, memory utilization are expected to be checked in order to see if there is the need for the expansion of the system or the other way round, so as not to waste resources.

2.2 Messaging Gateway

Messaging gateway synonymously called SMS gateway is a tool which allows exchange of SMS messages between two or more Short Message Service Centers (SMSCs) that support different protocols. Some of these protocols include: Computer Interface to Message Distribution (CIMD) for nokia, External Machine Interface (EMI) for another vendor called CMG Wireless Data Solutions etc. However, these gateways can be used to translate one protocol to another, send/receive SMS between mobile phones as well as between mobile phone and a computer etc.

SMS Centre (SMSC) handles the management of SMS like sending SMS messages to the appropriate destination, storing the message when recipient is unavailable and forwarding it to the recipient as soon as he/she becomes available. Developer's home (2010) reveals that, when

an SMS message is sent from a mobile phone it will reach an SMS Centre first, the SMS Centre then forwards the SMS message towards the destination. An SMS message may need to pass through more than one network entity (e.g. SMSC and SMS gateway) before reaching the destination. Figure 2.1a shows how SMS is sent and received between SMSC supporting same protocol, while Figure 2.2b shows how SMS is handled between SMSC supporting different protocols.

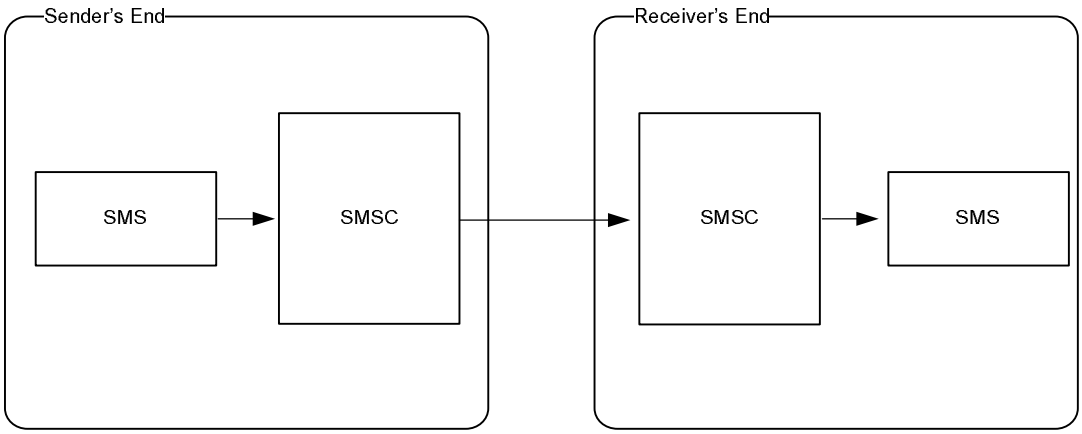


Figure 2.1a: Handling SMS that support same protocol

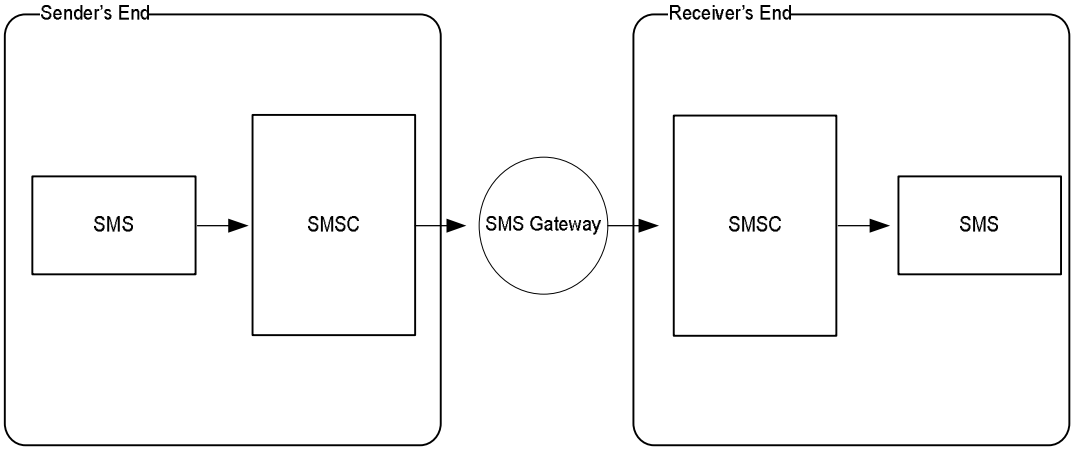


Figure 2.1b: Handling SMS that support different protocol

Sending an SMS requires a user to know the SMS service provider's address, which is normally a phone number in an international form. Even though, there is an option for configuring this address in a mobile phone but the address is already configured in the SIM card by the operators.

Different vendors have developed their SMS gateway, some commercial and some free of charge. The choice of which SMS gateway to use depends on the functionality you want; you need to check some and choose from. Some of these SMS gateway include Gammu, NowSMS, Ozeki NG, MCTEL, ZONITH etc.

According to ZONITH (2007), ZONITH SMS Gateway is a server solution that enables both local and remote applications to send and receive SMS messages in real-time. The communication with SMS Gateway is based on XML messaging (SOAP), emails (SMTP or POP3), and/or SMS messages. This SMS gateway is divided into client and server software, the server software works only on Windows platform while the client works on Windows, Linux and UNIX. The features supported by ZONITH include notification services (message sent to group on subscription), text messages (SMS), WAP PUSH messages (message sent to users phone and a URL to follow a link), flash message, email. You can also setup the gateway to use multiple GSM modems in order to have non-breaking message delivery in case of a modem failure.

MCTEL (2010) describes their gateway as a message gateway that allows Value Added Service Providers (VASP) and Corporate organizations to connect to the Mobile Network operator gateways in order to manage SMS and offer SMS Value Added Services and applications. This gateway supports message forwarding or exchange using optional connections such as: FAX,

MMS, TELEX, EMAIL, Voice text to speech, etc. It also has support for C/C++, php scripts, HTTP/HTML.

Ozeki NG (2010) reveals that, Ozeki NG SMS gateway support high capacity SMSC connections over SMPP, UCP, CIMD2 or GSM Modems. It can be used over HTTP SMS connections. It is an ideal platform for SMS notification, SMS paging, two way SMS systems, or for becoming an SMS service provider. It supports SMS API for SQL, .NET, ASP, ASP2, VB, C#, C++, Delphi, PHP, JAVA and for OEM vendors. The gateway has the capability for sending and receiving up to 400 SMS messages per second (tested on an Intel P4 PC), and allows up to 1000 simultaneously connected applications.

According to Now SMS/MMS Gateway (2010), NowSMS is an SMS and MMS Content Delivery Solution. It has features like operator independent WAP Push support, conversion for any MMS content, bi-directional e-mail to MMS gateway, 2-way SMS and MMS application support, etc. The gateway is designed for windows XP, Vista, 2000, 2003 and 2008.

Chow (2009) describes Gammu as the most popular program to build customized SMS server. It can connect to several handsets, from the oldest model until recently launched models. Also it supports popular operating systems such as Microsoft Windows and Linux Ubuntu. Gammu is a command line utility written in C language, which can be used to control mobile phone. This tool provides access to so many phone features, some of these features include:

i. Call listing, initiating and handling
ii. SMS retrieval, backup and sending
iii. MMS retrieval

iv. Phonebook listing, export and import (also from standard formats such as vCard)

v. Calendar and tasks listing, export and import (also from standard formats such as vCalendar or iCalendar)

vi. Retrieval of phone and network information

vii. Access to phone file system (note that some phones work also as USB storage devices and those are not accessible through Gammu) (Cihar, 2010).

The Gammu package is a collection of binary files, Gammu SMS Daemon, Gammu library and Python bindings can be used to develop robust mobile phone applications. However, in order to know the phones and features supported for each phone by Gammu library, you need to refer to Gammu phone database. The supported phone models include: Alcatel, Apple, Huawei, LG, Motorola, Nokia, Sagem, Sony Ericsson, Samsung, etc. Among these manufacturers different models are supported.

Gammu supports several commands for manipulating mobile phones, most of the commands include several options. The general syntax for executing Gammu command is

Gammu [--]command [options] , e.g. Gammu identify.

The -- and options are optional.

We have answercall, cancelcall, dialvoice, divert, holdcall, etc for commands related to making and receiving calls. Commands like savesms, getsmsm, addsms, backupsms, sendsms, setsmsc, etc, are used in sending and receiving SMS messages. For memory (phonebook and calls), we have commands like addnew, addcategory, backup, clearall, getmemory, searchmemory, restore, savesms, savefile, etc. We also have various commands for pictures, ringtones, calendar, to-do lists, notes, date, time, alarms, network, WAP, radio, backup and restore, etc (Cihar, 2010).

2.3 SMS

Short Message Service (SMS) is a method of communications which allows the exchange of short text messages between mobile phones or other electronic devices. This is one of the vital services provided by mobile GSM network. The "short" part refers to the maximum size of the text messages: 160 characters (letters, numbers or symbols in the Latin alphabet). For other alphabets, such as Chinese, the maximum SMS size is 70 characters. The idea behind inventing SMS was to overcome a communication problem like out of coverage or user's mobile phone turned off (Hord, 2005).

Developer's Home (2010) reveals that, SMS first appeared in Europe in 1992. Initially SMS was included in the Global System for Mobile Communications (GSM) standards but later it was carried to other wireless technologies like Code Division Multiple Access (CDMA) and Time Division Multiple Access (TDMA). The GSM and SMS standards were in the first place developed by ETSI (European Telecommunications Standards Institute), but today the 3GPP (Third Generation Partnership Project) takes responsibility for the development and maintenance.

SMS is one of the successful technologies in the area of wireless networking; about billions of SMS messages are sent everyday due to some of these reasons:

i. SMS messages can be sent and read at any time

ii. SMS messages can be sent to offline mobile phones

iii. SMS messages are less disturbing than receiving or making calls

iv. SMS messages are supported by 100% GSM mobile phones

v. SMS is a suitable technology for wireless application to build on, etc.

Some of the applications of SMS messaging include

i. Person–to-person text messaging

ii. Provision of information like news, weather report etc. Using reverse billing as a method of charging the user.

iii. Since SMS messages can carry binary data, objects like ringtones, wallpapers can be encoded in SMS message for download

iv. Alerts and notification

v. Two way interactive text messaging applications, like search system.

vi. Remote system monitoring, etc (Developers' Home, 2010).

2.4 J2ME

Java 2 Micro Edition (J2ME) is a runtime environment designed to develop Java software for consumer electronics and embedded devices. Sun Microsystems started the development of J2ME, but is now supported by some of the biggest consumer electronics and embedded device manufacturers in the world (White and Hemphill, 2002).

Applications developed using J2ME can be used to control mobile phones, pagers, PDAs, home appliances, wristwatches, etc, or provide some sort of services supported by aforementioned devices.

Generally Java programming language has the ability to be written once and run anywhere (platform independent), which makes commercial end users to pick an interest in it, so as to reduce software development cost. Sun quickly expanded the scope of the java platform to meet the needs of developers after the first edition of Java (Java 1.0 and 1.1). This expanded platform included a set of user interface libraries together with an array of features for distributed computing and improved security. When Sun released the first edition of the expanded platform called Java 2 platform, they split it into three parts namely: J2SE, J2EE and J2ME. The targeted market for the three platforms above is shown in Figure 2.2.

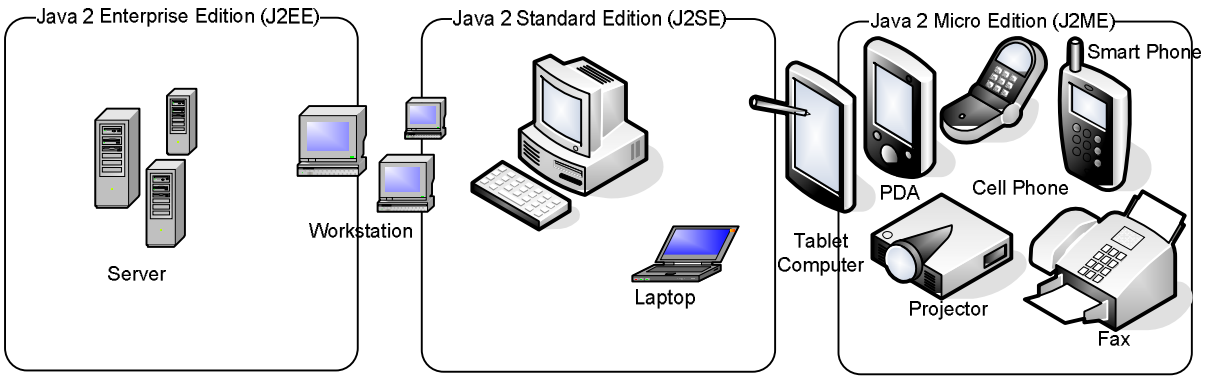


Figure 2.2: Platforms and targeted market.

J2ME is mainly divided into configuration, profile and optional packages, as shown in Figure 2.3. A configuration is a specification or set of low level APIs that defines the software environment for a certain class of device; the specification may rely on some characteristics like:

i. The amount of available memory.

ii. The kind of Java Virtual Machine (JVM) used.

iii. The type of network connection.

Two configurations are defined in J2ME, these are:

Connected Limited Device Configuration (CLDC)

This configuration is for small wireless devices with a limitation in memory, processing power and network connection. Typical devices that support this configuration include: cell phone, pagers, PDA, etc.

Connected Device Configuration (CDC)

This configuration is a subset of Java SE; it is richer than CLDC since it has almost all the non-GUI libraries for Java SE. The devices that this configuration supports have more memory, more reliable network connection and more capable processors. CDC might be found on high-end PDAs and in smart phones, web telephones, etc.

A profile builds on a configuration by adding additional classes that provide features appropriate to a particular type of device. Profiles include APIs for user interface, persistent storage, event handling, etc. Some of these profiles include:

Mobile Information Device Profile (MIDP)

This profile is designed for mobile phones that primarily aimed at the limited display and storage facilities. The MIDP was the first completed J2ME environment; it is based on CDLC with additional things like local storage, networking, and support for user interface. MIDP applications are called MIDlets.

PDA Profile (PDAP)

This profile is similar to MIDP, but it addressed the needs of PDAs in especially areas of user interface capabilities such as touch screen, event handling and data storage (Topley, 2002).

Information Module Profile (IMP)

IMP is a profile created for embedded devices; these devices have simple or no display. Example of embedded devices include: parking meters, emergency call boxes, security systems, etc User Interface API, game API and lcdui package are not supported in this profile. IMP applications are descendent of MIDlet class, so they have the same life cycle, packaging and deployment procedures; but unlike MIDlet they are called IMlet (Oracle Sun Developer Network, 2005).

Game Profile

The Game Profile is designed to handle the problems associated with games on mobile devices. The following features will be covered in this profile:

i. Multi player game oriented features like joining game, real time multi player

ii. Vector graphic operations in game

iii. 3D Engine

iv. Game Controllers

v. Sound for Games, etc (Oracle Corporation, 2011).

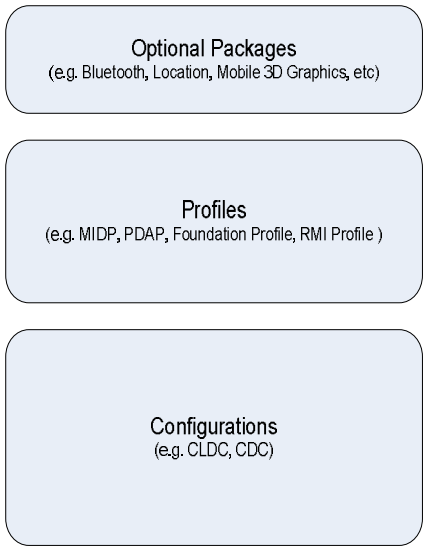


Figure 2.3: J2ME configuration, profiles and optional packages

The programming environment for consumer electronics consists of Java Virtual Machine and collection of Java classes. CLDC runs on a type of JVM called Kilobyte Virtual Machine (KVM), which is a portable Java Virtual Machine designed for devices with limitation in processor and memory. While CDC Virtual Machine or Compact Virtual Machine (CVM) is a virtual machine that supports all the characteristics of JVM, it is the virtual machine for CDC devices (White and Hemphill, 2002.)

2.5 WMIC

Windows Management Instrumentation (WMI) is a powerful technology for Microsoft's systems management; the system can be personal computer or a server. The advantage of WMI is that an administrator does not need to be a programmer to derive its benefit. Microsoft developed a command line tool called Windows Management Instrumentation Command-line (WMIC) for easy execution of WMI pre-defined and user defined queries. Since Windows XP, WMIC is part of the default installation of Windows operating system; even though WMIC is not part of

Windows 2000, but you can remotely query it from other machine that supports it (Lynch, 2006). According to Wilansky (2002), WMIC command line components include global switches, aliases, verbs, etc. Global switches are settings applied to WMIC session. For example, the /OUTPUT:test.txt sends an output to test.txt file, the /NODE switch lets you specify a machine to operate on. The /INTERACTIVE:on switch sets the interactive mode e.g. prompt Y/N before deleting a file. Other global switches include /USER for login session, /APPEND for output redirection mode, and /AUTHLEVEL for client level authentication.

Aliases are keywords defined to represent some component of a computer, e.g. BIOS, CDROM, CPU, LOGON, NETUSE, UPS, NIC, etc. These are used for management of devices/services corresponding to their names. The actions taking while specifying an alias is represented by Verbs, e.g. GET, CREATE, DELETE, CALL, etc (Wilansky, 2002).

You can access class directly or create its instance using CLASS command, PATH is also a command that is used to access an instance directly without using an alias. Other commands include, CONTEXT (to show state of all the global switches), QUIT and EXIT (for getting out of WMIC command prompt).

There are two ways for using WMIC, the interactive and batch mode. To run WMIC in interactive mode, simply type WMIC in the windows command prompt, a WMIC command prompt will be initiated as shown in Figure 2.4.

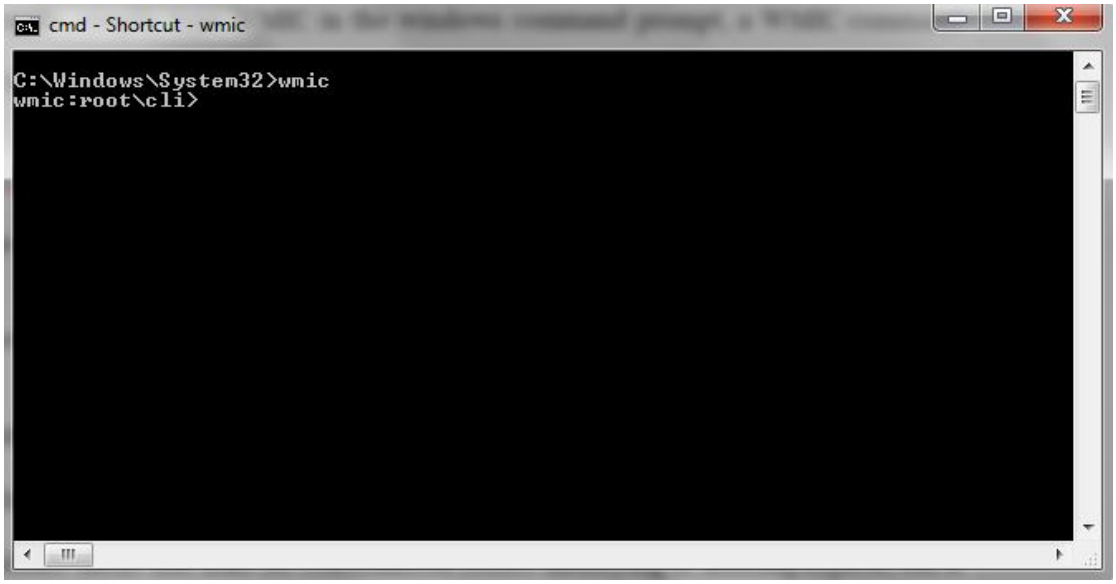


Figure 2.4: WMIC specifying command in interactive mode.

For example to close a wordweb dictionary which is an application currently running, you can issue a command

process where name='wordweb.exe' call terminate

Running WMIC in batch or non-interactive mode requires you to type WMIC and the commands in the same line, e.g. wmic useraccount list brief. However, the interactive version sometimes asks for confirmation before performing an operation. Generally, the syntax for issuing command in non interactive mode is:

wmic [credentials] [area] [querystring]

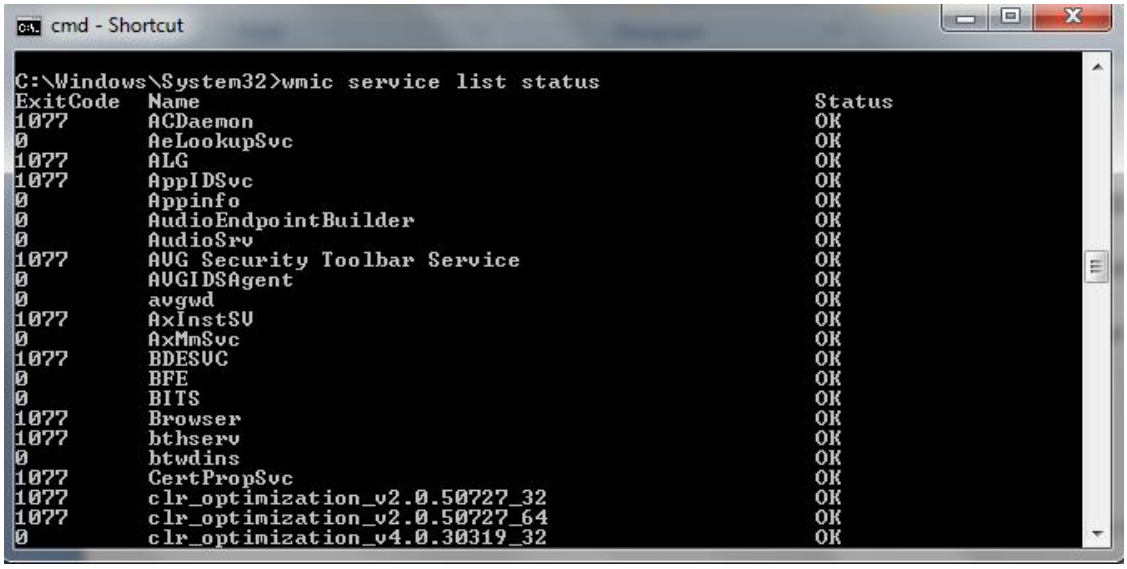


Figure 2.5: WMIC specifying command in batch mode

Figure 2.5 shows a command for listing the services on a computer with their status (Lynch, 2006).

WMI QUERY Language (WQL) is a language similar to SQL for manipulating WMI. Apart from the predefined queries like *wmic cpu get*, WQL can be used to create user defined queries.

2.6 Wake On LAN

Wake On LAN (WOL) is a technology used to wakeup systems, by sending a special network packets called magic packet to machines enabled to respond to the packets. This feature eases the work of administrator in the sense that, thousands of computers can be turned on in seconds rather than visiting all the computers and press start button.

Lieberman Software (2006) reveals that, the WOL technology came from the Intel and IBM Advanced Manageability Alliance formed in October of 1996. This alliance introduced the Wake on LAN technology in April of 1997. Other industry initiatives such as Wired for Management (WfM) provide support for Remote Wakeup via Wake on LAN.

To be able to wake a computer using Wake on LAN; the network adapter must be receiving power even if the computer is turned off, BIOS must be configured to support WOL in power option, also if the network adapter is outboard WOL cable must be connected to the motherboard.

Magic packet is a broadcast frame that consist of 6 x 0xff (ffffffffffff) followed by target computer's MAC address (48-bit) repeated sixteen times. The packet is typically sent as UDP to address 255.255.255.255 on port 7 or 9, but can be on any port. For example,

FF FF FF FF FF FF 00 0D 61 08 22 4A 00 0D 61 08 22 4A 00 0D 61 08 22 4A 00 0D 61 08 22 4A 00 0D 61 08 22 4A 00 0D 61 08 22 4A 00 0D 61 08 22 4A 00 0D 61 08 22 4A 00 0D 61 08 22 4A 00 0D 61 08 22 4A 00 0D 61 08 22 4A 00 0D 61 08 22 4A 00 0D 61 08 22 4A 00 0D 61 08 22 4A 00 0D 61 08 22 4A 00 0D 61 08 22 4A. (AMD, 1995).

2.7 Related Work

A Java based network monitoring system was developed by Xi (2000) in order to monitor and control a network. Some features of Java technology were taken as an advantage and presented as a framework for distributed and dynamic network monitoring. Specialized Java objects known as Intelligent Monitoring Objects were delegated to a Java based Extensible Management Server (JEMS), where they carried out encapsulated monitoring functionality upon management information collected locally from the underlying network device. These objects automate the monitoring and analysis of corresponding network. However the main weakness of this application is lack of security and mobility. Authenticating a user with username and password is what will make the system to some extent secured, while the lack of mobility makes the user to

be with his/her PC to monitor the server. We intend to design and develop a system that overcomes both of these limitations.

Jaoude *et al* (2004) described the architecture of a secure home-management system which can be remotely controlled through a common mobile device. A General Packet Radio Service GPRS subscription allows the mobile to communicate with the home-server via the Internet which creates the necessary link required for accessing the control hardware, or alternatively Short Message Service (SMS) provides such communication. The project was divided into three major categories: the Client (or Remote side), the Server (or central segment), and the Hardware (or Device manipulation). These three main components are linked to each other with the appropriate communication channel. The mobile phone (client) communicates with the web-server through two media. First, data signals are sent to the GSM provider's gateways using GPRS. These are then routed to the public Internet and eventually to the household server via the second medium which is the Transmission Control Protocol/Internet Protocol (TCP/IP). The server communicates with the control hardware using the RS232 DB9 COM port on the computer which is a serial connection. Each stage of the project is developed in order to communicate (send/receive) and perform the required tasks efficiently. The main difference between this project and our proposed project is, in this you must have GPRS enabled phone and service to manage the devices. The SMS alternative serve only the purpose of notifying the user if something happens to the devices, but in our proposed system we don't use GPRS since its not as common as SMS service especially in our localities.

Liu (2008) developed remotely controlled mobile application for horticulture. His work was aimed at developing an integrated database and GUI (Graphical User Interface) for typical 3G, or

2.5G, mobile phone with Java MIDP2 capability. He investigated four embedded databases based on J2ME; also user interface software like J2ME Polish for mobile phones has been investigated. To provide a graphical presentation of data collected, a custom J2ME class for graphical interface was developed.

Litoriya (2009) developed SMS based home control system called SBHC using the GSM network, whereby the system allows monitoring and controlling of home appliances such as microwave, lamp *etc* through mobile phones by sending a command in the form of SMS messages. The authentication in this system is done at the backend, unlike our proposed system which is done at both ends.

Liaw and Ou (2003) implements a remote monitoring platform on the wireless network by using Java 2 Micro Edition technology (J2ME) and Java 2 Enterprise Edition (J2EE), named "Mobile Information Service System". This system was separated into the Server-side and Client-side. A device for capturing an image was used to capture the state of the server at the server side; the image capturing application was implemented using Java Media Framework (JMF) technology. The client's wireless application was developed using Mobile Information Device Profile (MIDP) package. Meanwhile, the user needs to download the application to their mobile phones so that, they can monitor the states of the servers as well as having access to the additional service provided by the system which is Online Dictionary. This brings about additional overhead for the system, because a camera has to be connected with the system, and also processing the image requires extra time.

Alomari (2008) developed a prototype for university mail notification called "Universiti Utara Malaysia (UUM) Mail Notification System Using Mobile SMS Application" using Asp.net with

the dot net framework; the database server used was Microsoft SQL Server 2005. OZEKI NG-SMS gateway was chosen as the SMS gateway connected with Nokia N95. The system sends SMS containing a brief description of the mail (letter, bills and parcels) to student's mobile phone, notifying him/her about the received mail rather than the manual way of going to the block mailbox and check if they have mail. The system comprises of three users, these are: administrator, employee and student.

Administrator: Responsible for managing employee, residential college and updating the application.

Employee: Responsible for managing student, send SMS to students for mail receive notification or daily reports on the number mails received.

Student: Responsible for managing his/her details, can select type of SMS notification required, can check details of a mail received online.

The system is not interactive, because the user cannot send request (enquiry) about received mail, he can only receive notification upon receiving a mail.

Sufi et al (2006) proposed a generic smart telemonitoring platform. In this system a Java based software running on the mobile phones perform computation intensive tasks like biosignal analysis, compression, encryption, etc. The system automatically alerts medical service provider, when medical assistance is crucial for the user. The proposed system contains five main subsystems. These subsystems are: patient's unit, doctor's unit, web server, database and web mining agent. Patient's unit contains biosignal acquisition device with Electrocardiography

(ECG), Oximetry, skin impedance and blood pressure sensors connects to the mobile application running on the patient's mobile phone through Bluetooth streaming.

Doctor's unit is a mobile application running on the doctor's mobile which notifies the doctor in case of emergency, reporting and analyzing the patient's condition is also done at this unit. Web server unit provides information to authorized medical personnel. Database unit keeps the biological signals stored using SQL Server 2000. Web Mining Agent unit is responsible for supplying information to the doctor's mobile phone; this unit makes the information requested by the doctor more readable.

The system provides the following functionality: SMS notification, regular signal submission, real time monitoring, detection algorithm, compression algorithm, etc.

Sullivan (2004) in his work explores various methods of remote network administration for both wired and wireless networks in an attempt to provide solution for secure remote network administration in Department of Defense. Telnet, SSH, SNMP, remote desktop, etc were discussed as network management utilities. Remote power management options like remotely managing UPS with automatic reboot technology were discussed.

Cosenza (2005) describes Enterprise Messaging Gateway (EMG) as a commercial SMS gateway developed at Infoflex (a Swedish Company) that interconnects SMS Centers (SMSCs) providing routing and message conversion services. Cosenza (2005) worked on extending the capability of EMG in two ways. The first was extending the monitoring capability by designing and implementing a new Management Information Base (MIB) using SNMP framework, which will offer EMG managers a standard interface for the monitoring. The second work was providing

congestion control capabilities using a policy-based approach. Before this work, network managers have to write a script in order to get information through command line EMG utilities, the system also lacks congestion control capabilities.

Wu and Jan (2003) proposed a system called Home Network System (HNS), which is an integrated system that remotely monitor and control devices connected to it via GSM mobile terminal or a laptop computer. The system comprised of HNS gateway and three home network subsystems (home appliance, security and messaging subsystems) as part of its architecture. The managed devices can automatically send message as an alert to a mobile terminal when an abnormal state occur, likewise the devices respond to queries sent by sending message back to the user. The home appliances subsystem do not have management services at the time of implementation, rather a simulation program have been implemented using PCs and their peripheral to emulate the appliances. In the security subsystem charged-coupled device (CCD) cameras connected to a PC was used for the surveillance system. While in the messaging subsystem only email services are provided. The HNS gateway was implemented on a desktop PC and a GSM mobile station connected through RS232 serial port for sending and receiving SMS messages. The software involved in HNS gateway includes Web server, WAP gateway, remote access service (RAS) server, short message driver and HNS configuration tool. A modem is required since a laptop computer or WAP GSM mobile station will need to communicate with the gateway. The HNS configuration tool is a management tool for locally monitoring, controlling and coordinating a various set of independent appliances connected to the HNS.

Maxim Integrated Products (2004) demonstrates how to make a simple application that remotely monitors temperature and analog voltages. The target of the system is to be able to send email

whenever certain alarm condition is met, but the system first sends SMS message using GSM module to a tool known as TynTec that convert the SMS to email. The system components are PIC microcontroller (PIC 16F877), 1-Wire temperature sensor (DS18S20), 8-channel 12-bit serial A/D converter (MAX127), GSM modem and SMS to email conversion service provided by TynTec.

Da'na *et al* (2008) developed a platform to remotely monitor and control Programmable Logic Controller (PLC)-based processor. PLC is microprocessor-based control system that can be used to automate electromechanical processes; these processes include control of machinery in automobile industry, production industry, etc. A control program stored in the PLC memory determines the relationship between the inputs and outputs of the PLC. The components of the system include Simatic s7 200/300 PLC systems and Communication Processors (CPs), connectivity between the process and client/administrator via network, a database server and a variety of network options including GSM-based accessibility. The PLC system can be accessed through GSM network by the Administrator or a client. SMS capability is also incorporated into the system, where by the administrator can be able to send a request to the database server or PLC and an SMS message can be automatically sent to the administrator periodically or in case of emergency.

Mobile bicycle monitoring system is a system of collecting information generated while cycling presented by Honkasuo *et al* (2006), the aim of the system is to collect some data from the bicycle sensor and sending it to the mobile phone for analysis as well as storing the location of the user temporarily. The system is divided into two parts, bicycle sensor module and the mobile application on the phone. The communication between these two parts is via Bluetooth wireless

technology. The Bicycle sensor module uses heart rate, pedal and tire sensors and is powered by rechargeable batteries that gains energy from the bicycle dynamo. This model sends the amount of pedal rotations, tire rotations and heart rate information to mobile phone, where by the application running on Symbian OS on the mobile phone calculates speed values, time intervals, different distances, pulse values and calory burn. These data can be transferred from the phone to PC for long term storage and further analysis. The system is deficient in terms of security, since the connection between the sensor and mobile phone is not secured.

2.8 Conclusion

This chapter talks about management of servers and messaging gateways, several types of gateway with their features were discussed. SMS, J2ME, WMIC and Wake On LAN technology were also looked into. Finally on this chapter literatures related to our work were reviewed in order to have better understanding of the concept.

CHAPTER THREE: DESIGN AND IMPLEMENTATION

This chapter looks into the design and implementation of the system; it also describes the system architecture. Java 2 Micro Edition (J2ME) is a Java based technology for developing mobile application together with batch file programming for processing the messages were used for the implementation.

3.1 System Architecture

The architecture of the system as illustrated in Figure 3.1 is divided into two parts: Frontend and Backend. The Frontend is a mobile phone equipped with an application developed using J2ME technology for sending Short Message Service (SMS) to the backend. On the backend there is a server that will be monitored and a machine that will monitor the server. The machine that will monitor the server has a mobile phone attached for receiving SMS and a database for storing the messages received or messages that would be sent, there are also batch files (files containing commands to be processed in batches one after another) for automating the process of checking the status of the server, services on the server, as well as handling the receiving and sending of SMS. Gammu was chosen as our SMS Gateway because of the following benefits:

i. It is command line based.

ii. It is free software.

iii. Has support for so many mobile phones.

iv. Can be installed on Windows, Linux and Mac OS X platform.

v. Can write online or offline applications with it.

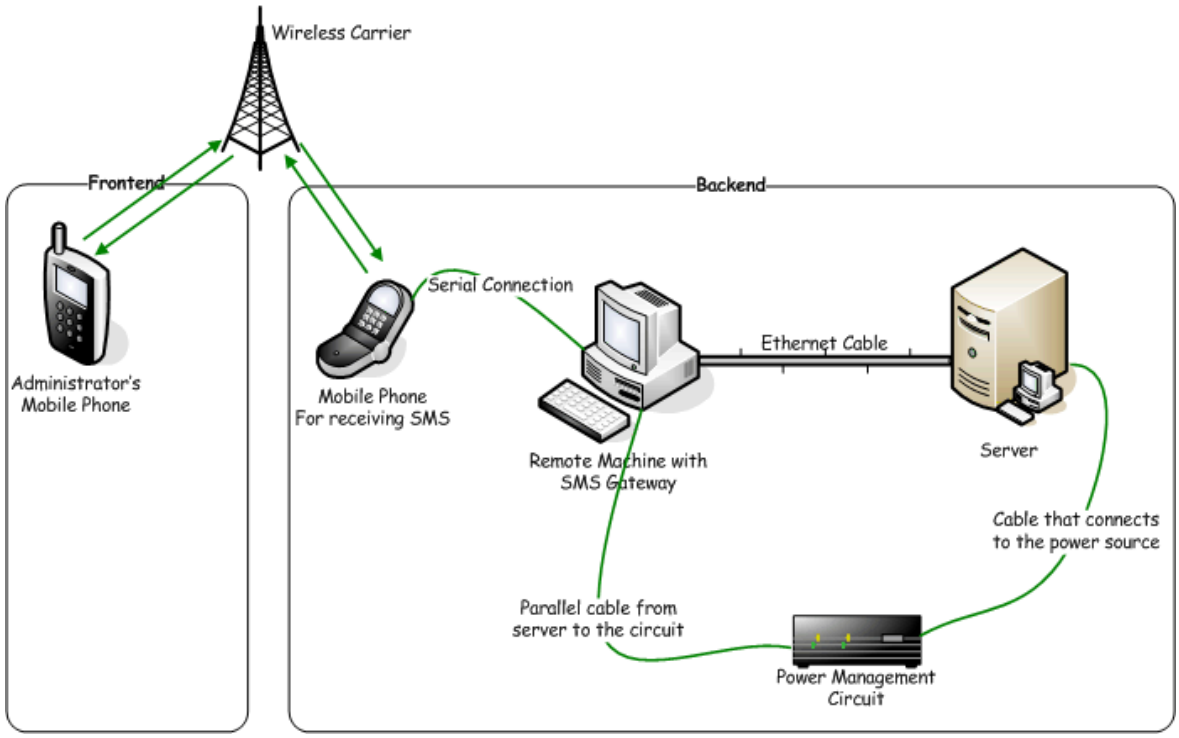


Figure 3.1: General system architecture

3.2 Implementation

This section discusses the architecture of the complete system, which includes the frontend and backend.

3.2.1 Frontend Architecture

In this research, we use Mobile Information Device Profile (MIDP) to implement the mobile phone's messaging application, which will serve as the frontend. This application will be used by the administrators for sending a request to the computer that monitors the server in the form of a message.

The Connected Limited Device Configuration (CLDC) allows a programmer to write an application that is intended to be run on devices with limited resources like, CPU, memory etc.

The CLDC API includes packages as: **java.lang**, **java.io**, **java.util** and **javax.microedition**. The first three packages are from Java 2 Standard Edition (J2SE) but not all methods implemented in J2SE are implemented here (eg finalize()). However, developing a mobile application often requires interaction with users, which is lacking in CLDC API; this brings about creating a profile that will provide the required functionality apart from those provided by CLDC.

MIDP is one of the Java mobile profiles that works on top of CLDC. A MIDlet is a Java class derived from an abstract class **javax.microedition.midlet.MIDlet**, the methods **startApp()**, **pauseApp()**, and **destroyApp()** are implemented by a MIDlet. Generally, an MIDP application includes other classes with some resources apart from the main MIDlet class that inherits **javax.microedition.midlet.MIDlet**, these are collectively known as MIDlet Suite.

A MIDlet is always in one of the three states during its life cycle: **Active**, **Pause**, or **Destroy**. When a MIDlet is loaded, it is in paused state initially and initialization for class and instances is done at this stage. If at this stage during executing the constructor of the MIDlet throws an Exception, its then destroyed; otherwise it is changed from Paused to Active state by invoking its **startApp()** method.

There might be some cases when a MIDlet is running (in Active state) and need to be paused e.g. if a user received a call, in that case a **pauseApp()** method is invoked and put the MIDlet in **Pause** state. The **startApp()** method is called again in order to resume to **Active State**.

The **destroyApp()** method is called to terminate a MIDlet, in this case all resources allocated to the MIDlet will be released. The figure below shows the state diagram for a MIDlet.

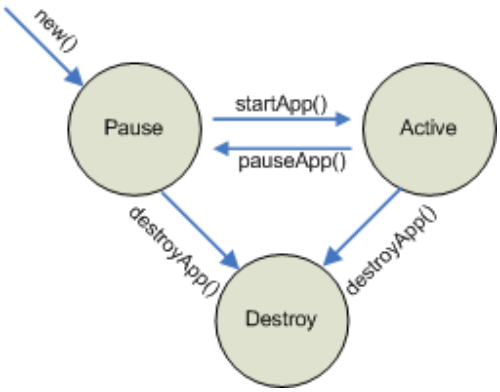


Figure 3.2: MIDlet life cycle

Developing a MIDlet is a seven stage process: Design, Code, Compile, Preverify, Package, Test and Deploy.

Sun Java Wireless Toolkit for CLDC version 2.5.1 (WTK2.5.1) is software that enables a programmer to develop, preverify and package MIDlet suites. The stages are briefly discussed below:

Design: In this phase the requirements are well planned, the packages and classes needed are also decided. Our aim now is to design a MIDlet for sending SMS, which will require a user to be authenticated before granting access to the application. We now chose the version of CLDC and MIDP in this phase, which are CLDC 1.1 and MIDP 2.0 respectively.

Code: This stage involves writing the code for the application. Java programming language was used for the development. NetBeans and JCreator are the Integrated Development Environments (IDEs) used.

The packages needed for the development include:

i. *javax.microedition.io* for managing connections (e.g. opening and closing a connection).

ii. *javax.microedition.lcdui:* This package provides set of features for implementing user interfaces for MIDP applications. Form, DateField, ChoiceGroup, Alert, Font, Screen, TextField are some of the classes defined in this package for manipulating the liquid crystal display user interfaces.

iii. *javax.microedition.midlet.MIDlet* for controlling the MIDlet such as creating, starting, pausing and destroying a MIDlet.

iv. *javax.wireless.messaging* for managing the sending and receiving of wireless messages. The API defined in this package is independent of the underlying protocol such as GSM SMS, CDMA SMS, etc. MessageConnection is an interface responsible for sending and receiving messages; the application opens MessageConnection by providing a URL connection string. This type of connection can be opened either in server or client mode; server mode can be used for both sending and receiving messages, while client mode can be used only for sending messages. In this work a client mode is sufficient, since we are only sending SMS through the MIDP application.

Most of the user interfaces and messaging functionality needed for this application are defined in **javax.microedition.lcdui** and **javax.wireless.messaging** packages respectively.

After coding, the source files are placed in the src directory as described in Appendix B.

Compile: The compilation is done by clicking a button in WTK2.5.1. To compile our project, click on **Open Project** to select the project you want compile as depicted in Figure 3.3a and 3.3b, then after loading the project click on Build button as in Figure 3.3c.

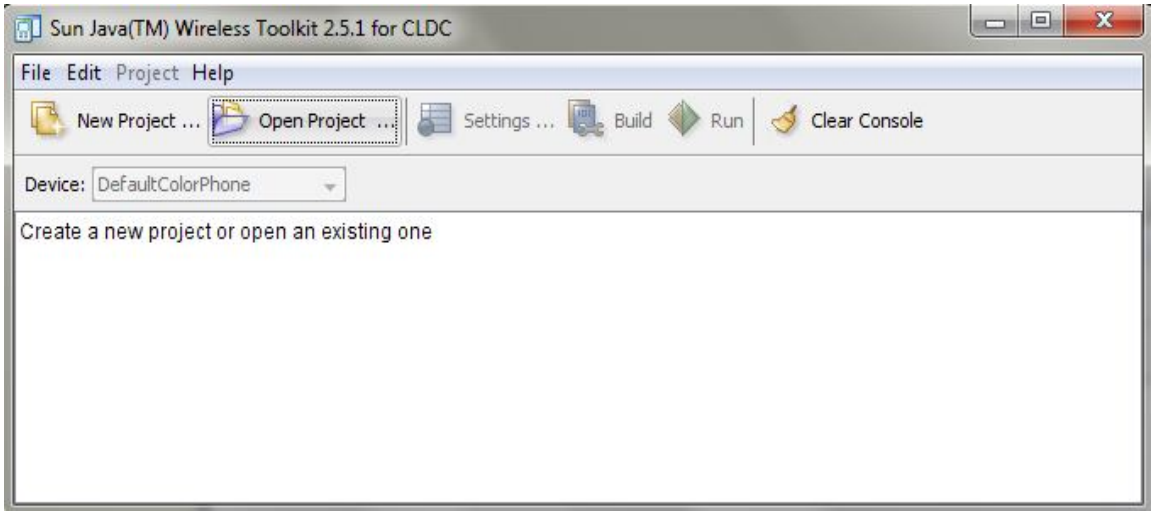


Figure 3.3a: Opening a project

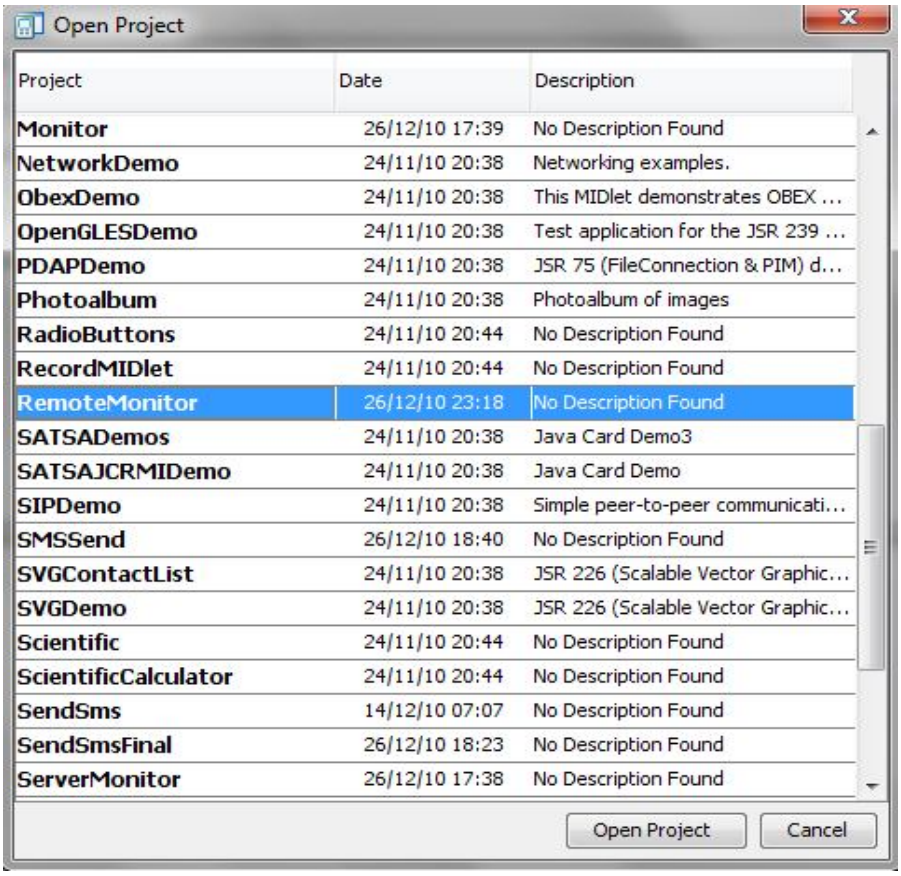


Figure 3.3b: List of created projects including sample from WTK developers.

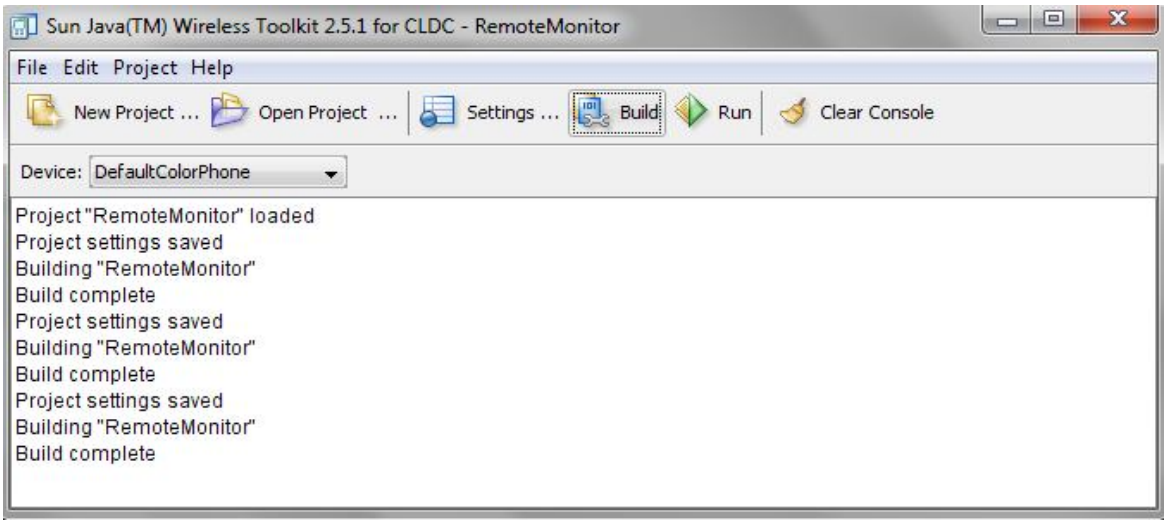


Figure 3.3c: Compiling a project

Preverify: This is done during the Build operation as shown in Figure 3.4. After clicking **Build** button, once you see **Build Complete** in the WTK console it means the project has been compiled and preverified successfully. Preverification is to make sure that the class file is structurally and conceptually correct as per the specification of the JVM.

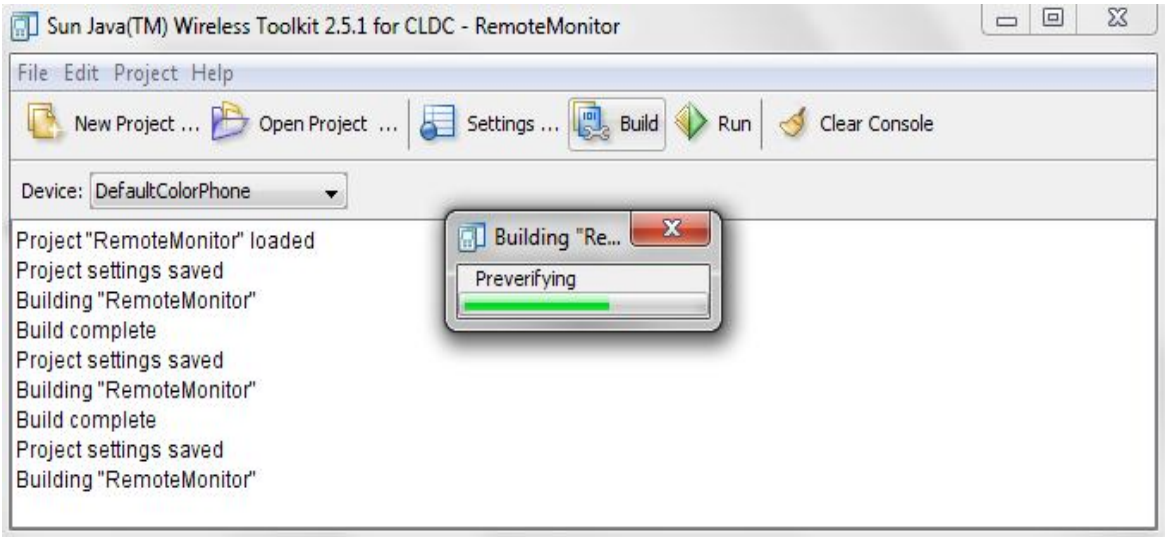


Figure 3.4: Preverification process

Package: In this phase, the MIDlet subclass together with any other classes, images or other files required at runtime must be built into a single file known as Java Archive (JAR) file. Packaging information like what is in a JAR file is supplied into a file called **Manifest**. A **manifest file** is a text file with **.mf** extension that has the information described above. Another file with similar packaging information called Java Application Descriptor (JAD) is also created. Table 3.1 shows a brief explanation of some JAD and Manifest files attributes.

Table 3.1: JAD and Manifest files attributes.

JAD File Attribute	**Description**	**Attributes example**
MIDlet-n	Attribute per MIDlet. Values are MIDlet name, optional icon, and MIDlet class name.	RemoteMonitor, RemoteMonitor.png, RemoteMonitor
MIDlet-Jar-Size	Size of the Jar file in bytes (optional).	10928
MIDlet-Jar-URL	Location of the JAR file.	RemoteMonitor.jar
MIDlet-Name	MIDlet suite name.	RemoteMonitor
MIDlet-Vendor	Name of the vendor who supplied the MIDlet.	Unknown
MIDlet-Version	MIDlet version number.	1.0
MicroEdition-Configuration	Configuration required to run the MIDlet.	CLDC-1.1
MicroEdition-Profile	Profile required to run the MIDlet	MIDP-2.0

Manifest File Attributes	Description	Attributes example
MIDlet-n	Attributes in a MIDlet, this includes MIDlet name, class name and icon.	RemoteMonitor, RemoteMonitor.png, RemoteMonitor
MIDlet-Name	Name of MIDlet suite	RemoteMonitor
MIDlet-Vendor	Vendor's name	Unknown
MIDlet-Version	Version number of the MIDlet	1.0
MicroEdition-Configuration	Configuration required to run the MIDlet.	CLDC-1.1
MicroEdition-Profile	Profile required to run the MIDlet	MIDP-2.0

Manifest file is like a subset of JAD file, since whatever is in Manifest file is in JAD file but not all what is in JAD file is in Manifest file. MIDlet-Data-Size is one of the attributes in JAD file that is not in Manifest file, the value for this attribute should be the actual size of the JAR file in bytes.

Test: The WTK2.5.1 comes with emulator for testing a project. Once you click on **Run** button in the WTK2.5.1 development environment; a mobile phone like emulator will be launched, thereby displaying the application like in a real mobile phone.

Deploy: To deploy the application, you copy the JAD and JAR files into the mobile phone. You can copy only JAR file and install in some mobile phones, but some phones require that you copy all the two (JAR and JAD) in order to install the application.

Meanwhile, today almost all mobile phones do have the capability for sending and receiving an SMS; why then do we need this application just to send an SMS? The reasons for not using the default messaging facility that comes with a mobile phone are simply:

i. Security: In order not to allow unauthorized users to send messages, whenever you launch the application, a form with username and password field appears. The user has to be authenticated before sending the message.

ii. Avoid wrong number input: The backend has an SMS gateway for handling messages received from the user connected with mobile phone. However, the mobile number need not be typed by the user, so that a message will not be sent to a number other than that of the gateway as a result of forgetting the number or wrong input.

3.2.2 Backend Architecture

The backend consists of two computers: the server and another computer that monitors the server. The computer that monitors the server has a mobile phone connected to it via serial port for receiving SMS from the user, SMS gateway (gammu), and a circuit connected via parallel port that is also connected to the server for managing the power as shown in Figure 3.1. The platform used was Windows for the machine that monitors the server and Linux (Ubuntu) for the server. However, in the former system a batch file containing windows commands was used for handling the request sent by the user in form of SMS, as well as sending message automatically in case of something happens to the server or any of its services; while in the latter we have **awk** language commands in a shell script file for getting the status of the services and users on the server.

3.2.2.1 Setting up Gammu SMS Gateway

The Gammu configuration files (gammurc and smsdrc) stated in Gammu installation steps discussed in Appendix A will be briefly discussed here. gammurc is a configuration file that gammu reads information concerning the connected phone and the type of connection from, in order to communicate with the phone; the configuration file can contain several sections each identified in square brackets e.g. [gammu], [gammu1]…. [gammuN], each section configures one connection setup.

You can provide a number as a parameter to gammu in order to use a particular section e.g. [gammu] has number 0, [gammu1] has number 1, etc or by default if you have more than one section gammu will try them all in numerical order. Some gammurc properties include:

Connection: a protocol which will be used to talk to the phone e.g. fbus, dlr3, mbus, dku5, at, blueobex, irdaat etc.

Port: The address of the phone which depends on the connection e.g. /dev/ttyS0, COM1 etc.

Model: the model of the mobile phone e.g. 3310, 5212 etc. This is normally used if gammu does not recognize the phone model.

Synchronize Time: This is used to set time from computer to phone during initializing a connection.

Logfile: Used to keep a record of communication in a log file.

The gammu configuration file for our project is given below:

[gammu]

port = com1:

connection = fbus

model = 3310

smsdrc is a configuration file for gammu short message service daemon (gammu-smsd). gammu-smsd is a program that periodically scans GSM modem for messages received, stores them in a specific storage defined in smsdrc and sends messages from the storage specified.

gammu-smsd has options like:

i. -i for installing the daemon as windows service

ii. -u uninstalling the daemon as a windows service

iii. -s starting smsd windows service

iv. -k stopping smsd windows service, etc.

For example to start the daemon you type *gammu-smsd –c smsdrc*

The gammu-smsd configuration file has at least two parts: [gammu] and [smsd]. The first part is the same as gammurc described above excluding the logfile parameter. The second part is the configuration for the daemon (smsd), you can include optional parts like [exclude_numbers] and [include_numbers] for message filtering. Nevertheless, gammu has options for storing messages in a file or database. The database supported includes MySQL and PostgreSQL, it has also

support for libdbi which provides access to wide range of database engines such as SQLite, MS SQL, Firebird etc, but libdbi works only on Unix platforms.

The parameters for smsdrc include:

Service: smsd service to use e.g. files (stores messages in files), mysql (stores messages in mysql database), null (does not store message), etc.

PIN: This is the PIN for the SIM card if it is set.

LogFile: A file where smsd actions are stored.

MaxRetries: How many times will smsd try to resend message if sending fails, the default is 1.

DeliveryReport: Used for keeping delivery report, either log, sms or no should be used.

PhoneID: This is a string used to represent phone used for sending and receiving SMS, used especially if running several daemon.

RunOnReceive: used for executing a program after receiving a message.

User: username used to connect to the database.

Password: the password used to connect to the database.

PC: the address of the database server.

Database: the database name to use.

For message filtering, if a number is used as part of include number, then that means only message received from this number will be processed. The sample smsdrc for our project is given as follows:

[gammu]

port = com1:

connection = fbus

model = 3310

[include_numbers]

number1 = 08063238793

[smsd]

Service = mysql

PIN = 1234

LogFile = smsdlog

User = root

Password =

PC = localhost

Database = smsd

debuglevel = 255

PhoneID = Barroon

RunOnReceive = C:\main.bat

The database used for this development was MySQL, the tables in the database include: daemons, gammu, inbox, outbox, outbox_multipart, pbk, pbk_groups, phones, and sentitems. The structures of tables for gammu MySQL database are shown below:

Daemon table: This is a table that holds information about running daemons; its structure is shown in Table 3.2.

Table 3.2: Daemon table structure

Field	Type	Null	Default
Start	Text	No	
Info	Text	No	

Gammu table: Table 3.3 below shows the gammu table that holds the version of the database schema.

Table 3.3: Gammu table structure

Field	Type	Null	Default
Version	int(11)	No	0

Inbox table: when messages are received, they are stored in this table. Its structure is shown in Table 3.4 below; the primary key for this table is ID.

Table 3.4: Inbox table structure

Field	Type	Null	Default
UpdatedInDB	Timestamp	No	CURRENT_TIMESTAMP
ReceivingDateTime	Timestamp	No	0000-00-00 00:00:00
Text	Text	No	
SenderNumber	varchar(20)	No	
Coding	enum('Default_No_Compression',	No	Default_No_Compression

Field	Type	Null	Default
	'Unicode_No_Compression', '8bit', 'Default_Compression', 'Unicode_Compression')		
UDH	Text	No	
SMSCNumber	varchar(20)	No	
Class	int(11)	No	-1
TextDecoded	Text	No	
ID	int(10)	No	
RecipientID	Text	No	
Processed	enum('false', 'true')	No	false

Outbox table: Table 3.5 shows the structure of outbox table that is used to keep messages enqueued for sending. ID is also the primary key for this table.

Table 3.5: Outbox table structure

Field	Type	Null	Default
UpdatedInDB	Timestamp	No	CURRENT_TIMESTAMP
InsertIntoDB	Timestamp	No	0000-00-00 00:00:00
SendingDateTime	Timestamp	No	0000-00-00 00:00:00
Text	Text	Yes	*NULL*
DestinationNumber	varchar(20)	No	
Coding	enum('Default_No_Compression', 'Unicode_No_Compression', '8bit', 'Default_Compression', 'Unicode_Compression')	No	Default_No_Compression
UDH	Text	Yes	*NULL*
Class	int(11)	Yes	-1
TextDecoded	Text	No	
ID	int(10)	No	

Field	Type	Null	Default
MultiPart	enum('false', 'true')	Yes	false
RelativeValidity	int(11)	Yes	-1
SenderID	varchar(255)	Yes	*NULL*
SendingTimeOut	Timestamp	Yes	0000-00-00 00:00:00
DeliveryReport	enum('default', 'yes', 'no')	Yes	default
CreatorID	Text	No	

Outbox_multipart table: This table is used to store data to be sent that are multipart, the first part of the mutipart messages is stored in outbox table. The SequencePosition field is used to keep the SMS number for linking with the first part. ID and SequencePosition are the primary key.

Table 3.6: Outbox_Multipart table structure

Field	Type	Null	Default
Text	Text	Yes	*NULL*
Coding	enum('Default_No_Compression', 'Unicode_No_Compression', '8bit', 'Default_Compression', 'Unicode_Compression')	No	Default_No_Compression
UDH	Text	Yes	*NULL*
Class	int(11)	Yes	-1
TextDecoded	Text	Yes	*NULL*
ID	int(10)	No	0
SequencePosition	int(11)	No	1

Pbk and Pbk_groups tables are currently not used by SMSD, they are only included for application usage.

Phones: This table stores information about connected phones, such as battery, signal level. The structure of the table is depicted in Table 3.7.

Table 3.7: Phone table structure

Field	Type	Null	Default
ID	Text	No	
UpdatedInDB	timestamp	No	CURRENT_TIMESTAMP
InsertIntoDB	timestamp	No	0000-00-00 00:00:00
TimeOut	timestamp	No	0000-00-00 00:00:00
Send	enum('yes', 'no')	No	no
Receive	enum('yes', 'no')	No	no
IMEI	varchar(35)	No	
Client	Text	No	
Battery	int(11)	No	0
Signal	int(11)	No	0
Sent	int(11)	No	0
Received	int(11)	No	0

Sentitems: Table 3.8 describes the structure of sentitem table, which is used to keep log of sent messages.

Table 3.8: Sentitem table structure

Field	Type	Null	Default
UpdatedInDB	Timestamp	No	CURRENT_TIMESTAMP
InsertIntoDB	Timestamp	No	0000-00-00 00:00:00
SendingDateTime	Timestamp	No	0000-00-00 00:00:00
DeliveryDateTime	Timestamp	Yes	*NULL*
Text	Text	No	

Field	Type	Null	Default
DestinationNumber	varchar(20)	No	
Coding	enum('Default_No_Compression', 'Unicode_No_Compression', '8bit', 'Default_Compression', 'Unicode_Compression')	No	Default_No_Compression
UDH	Text	No	
SMSCNumber	varchar(20)	No	
Class	int(11)	No	-1
TextDecoded	Text	No	
ID	int(10)	No	0
SenderID	varchar(255)	No	
SequencePosition	int(11)	No	1
Status	enum('SendingOK', 'SendingOKNoReport', 'SendingError', 'DeliveryOK', 'DeliveryFailed', 'DeliveryPending', 'DeliveryUnknown', 'Error')	No	SendingOK
StatusError	int(11)	No	-1
TPMR	int(11)	No	-1
RelativeValidity	int(11)	No	-1
CreatorID	Text		

3.2.2.2 SMS Handling

Batch files are chosen for handling the messages sent by the user. Batch file is a text file that has series of commands in a sequence that the user wants to be executed. Three file types were used for this part: namely batch (.bat), sql (.sql), and text (.txt).

The batch file manages everything, in the sense that it runs upon receiving a message and executes some batch files depending on what the user wants. It has the commands that ought to be executed, some of the commands used are:

Echo: This will print output to the user. Generally it prints to screen, but the output can be redirected to a file or other program segment.

Call: This is used to call one batch file from another.

Mysql: Used to launch MySQL database program.

@: This turns off the prompt from displaying the command specified, e.g. @echo off turns off the prompt permanently until it is turned on.

Begin: This is used to specify some actions that ought to be performed before processing records, mostly to set variables.

Rem: This is used to put comments/remarks.

Pause: This command suspends batch file processing until a user presses a key.

Set: Used to set an environmental variable, the environmental variable in DOS are normally global (can be accessible in all batch files).

Setlocal: Used to make environmental variable local within a batch file. The localization ends when an endlocal command is found or the end of the batch file is reached.

For: Used for running a specified command for each file in a set of files, or for each line in a file. It has the format *for %%var in (list) do*

If: Used for conditional processing of a program segment.

Goto: This is used to jump from one location to another in the script.

Find: This is used as a filter to find a line that contain or don't contain a specified string.

Environmental variables are specified with %% e.g. %%var1, but % is only used as in %var1 to define a command line parameter.

In some cases command output needs to be sent to a file or read input from a file known as redirection, < and > are the symbols used for reading input and sending output respectively. Generally, the syntax is command <in >out, this will read input from in-file and send to out-file. If the out-file does not exist, it will be created; otherwise the content is overwritten. To append an output to a file use >>.

The sql file contains the sql statements as an input to mysql command. The statements include:

Select: For selecting data from the database.

Use: To select a database or table for a particular operation.

Insert: For inserting record into a table in the database.

Load: For copying data from a file into a table in the database.

The idea behind using sql files is that, if you use mysql command in the batch file, the remaining commands that follow the line with mysql will not be executed until you get out of mysql terminal.

The text files are used to temporarily keep data for processing by the batch and sql files. These files serve as the input and/or output to most of the commands in the batch file.

3.2.2.3 Remote Command Execution

Generally, knowing the condition of services in a server involves running series of queries on the server; for example, to know if web server, email server or ftp server are running, you need to run some command in the server to get all the services running at that particular time and check the list for the service you are looking for. In the server we install Secure Shell (SSH) server that will allow us to remotely login into the server over a network; in the client which is the machine that monitors the server, we install PuTTY to enable us have access to the server remotely. We used some of the utilities provided by PuTTY that include PuTTY Link (plink) and PuTTY Secure Copy (pscp) in our work; plink gives you the flexibility for automating your processes by placing your commands either in batch file or shell script to be running non interactively, for example the following is the login command to the server having IP 192.168.0.22 with username barroon and password pass and also run a batch or script file called script.

plink barroon@192.168.0.22 -pw pass script

pscp is a tool for transferring a file from one computer to another securely, for instance the command below copies a file NUsers.txt from a computer(192.168.0.22) in StatusFiles folder to C directory of the client machine as NU.txt; -l is an option to specify the username and –pw for the password.

pscp -l barroon -pw pass 192.168.0.22:StatusFiles/NUsers.txt C:\NU.txt

Meanwhile, shell program (script) was used in the server in order to automate the process of checking the users as well as knowing the availability of any service on the server. According to Cooper (2010) a script supports almost all commands, utilities and tools for UNIX, and it has internal commands, such as loop and test construct that can be used to write a robust script. Creating a script file is not a big deal; you just create a file without an extension and write your commands in it. Some of the commands include: echo – for output, cp – for file copy, chmod – for changing file access permission etc; after creating a script you need to give read/execute permission to everyone on the file using *chmod 555 scriptfile* command, then to execute the file you type *./scriptfile*.

Awk is a script language for pattern scanning, Bezroukov (2010) describes awk as the most portable scripting language; it was created in late seventies, the name was from the initial letters of the three authors Alfred Aho, Peter Weinberger, and Brian Kernighan. The language has clean and simple syntax; it is usually used as a filter in pipes to reformat other command's output. Let us analyse a simple awk language statement:

who | awk '$5 !~ /:0/{print $1,$5}' | sort | uniq > StatusFile/Users.txt

who is a Linux command that shows who are logged on to a computer, ¦ represents a pipe, awk ' ' is the syntax, $5 !~/:0/ takes fifth variable if it does not contain :0, if the condition is true {print $1,$5} will print variable 1 and 5, ¦sort will arrange the result, and ¦uniq will give only one instance of the result if there are multiple instances.

3.2.2.4 Power Management

Engdahl (2009) describes parallel port as a 25 pin female connector that is normally used for connecting printer to a computer. Nowadays many hardware can be connected to computer

through this port, also the port can be used to connect electronic circuits to a computer for managing appliances. In this work, we have a circuit developed to switch on and off a computer using this port.

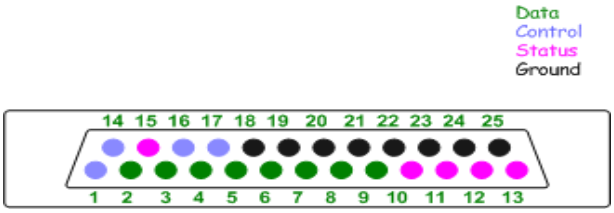


Figure 3.5: Parallel port pins

According to Tyson (2000), when a data is sent from PC through parallel port, 8 bits of data are transmitted in parallel unlike in serial where the data will be transmitted in a single row serially. When parallel port is used with printer each pin has a function, Table 3.9 shows a brief description of these pins.

Table 3.9: Parallel port pins and their function

DB25 Pin	Name of Signal	Register Bit	Direction	Function
1	-Strobe	C0-	Output	Set Low pulse >0.5 us to send
2	Data 0	D0	Output	Set to least significant data
3	Data 1	D1	Output	“”
4	Data 2	D2	Output	“”
5	Data 3	D3	Output	“”
6	Data 4	D4	Output	“”
7	Data 5	D5	Output	“”
8	Data 6	D6	Output	“”
9	Data 7	D7	Output	Set to most significant data
10	-Ack	S6+	Input	Low Pulse ~ 5 uS, after accept
11	+Busy	S7-	Input	High for Busy/Offline/Error
12	+PaperEnd	S5+	Input	High for out of paper
13	+SelectIn	S4+	Input	High for printer selected
14	-AutoFd	C1-	Output	Set Low to autofeed one line
15	-Error	S3+	Input	Low for Error/Offline/PaperEnd
16	-Init	C2+	Output	Set Low pulse > 50uS to init
17	-Select	C3-	Output	Set Low to select printer
18 – 25	Ground		-	

In our work we only use two pins, pin 2(D0) and 3 (D1) to send data from the computer that monitors the server to the circuit that connects to the server. The circuit as shown in Figure 3.6 performs the function of starting (on) and stopping (off) the server.

In the circuit D0 and D1 are diodes used to allow the flow of current in one direction, the current in the reverse direction is blocked by the diodes; at the same time D0 and D1 are the wires that connect to pin 2 and 3 of the parallel port respectively. To start the server we send data to D0, in other word we make D0 high, likewise to stop the server we make D1 high. In each case, when the data is sent it will first reach resistor that will reduce the flow of the current so that the current will not damage things like the transistor. From Figure 3.6, it is clear that the flow goes to the transistor where we have Base, Emitter and Collector wires, the base is what carries the data sent; the collector is the positive from 12 V DC sources, while the Emitter is the negative which connects the ground pins. For the relay to energize Collector (C) and Emitter (E) must come in contact, and for these two to be in contact there must be data from either D0 or D1; since we have data now they will be in contact. However, since the relay is energized, the magnet in it will attract one of the wires that go to the server's main power cable and make it in contact with the other that goes to the switch. With this, the system will be started and the wires in the relay that are in contact will be now be released, also when the system is working and data is sent through D1 the wires become in contact again for some time to shutdown the system and get released.

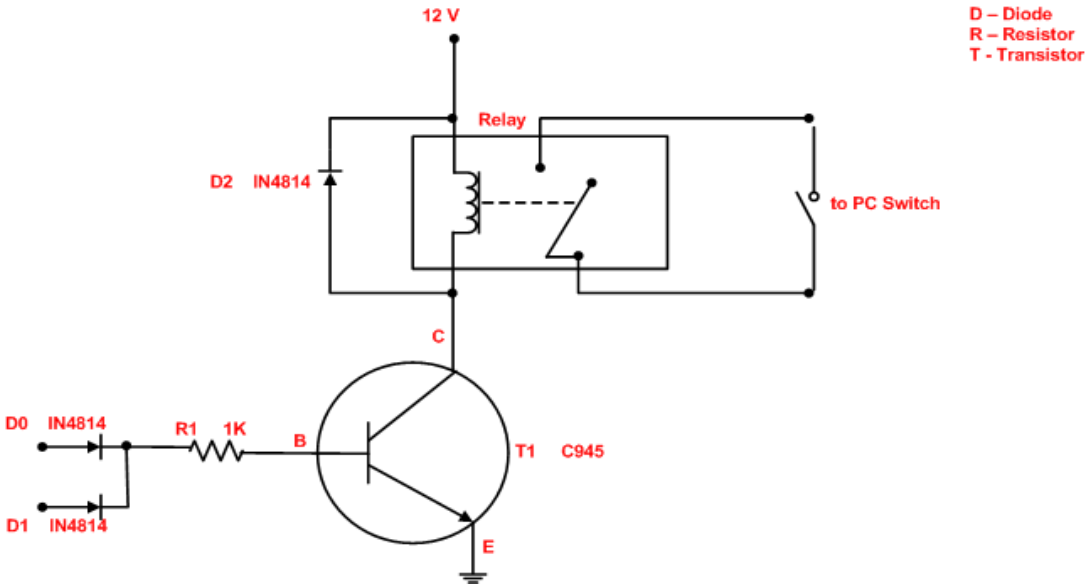


Figure 3.6a: Power management circuit



Figure 3.6b: Complete circuit snapshot

3.2.2.5 Tools Used

In this work, we used some tools developed by other researchers in order not to re-invent the wheel. Some of these tools include:

i. A batch file containing WMI script for checking battery status developed by Woude (2009). We make some modifications to the file to suit our need.

ii. An executable file (lptout.exe) for setting a particular pin to either low or high by Engdahl (2009). To run the program you navigate to the directory where you have the file through command prompt and issue the command, for instance *lptout 0* – set all data pins to low, *lptout 255* – set all data pins to high, *lptout 1* – set data pin D0 to high and all the rest to low, etc.

iii. We have another executable file (sleep.exe), that we used in the batch file when we want hold on executing the commands for a particular time. This is also command line based utility from Computer Hope (2010); to run simply type sleep 10 – to make the command line wait for 10 seconds before executing the next command.

iv. We also use PyTTY which is free client software for SSH and Telnet.

3.3 Conclusion

This chapter discusses about the architecture of the whole system, the system is divided into two: frontend that has the J2ME application on the user's mobile phone and backend that include various technologies. How to set up the IDE used for developing the J2ME application and the SMS gateway were described. A circuit for managing the power for the server was designed and illustration of how it works was reported. Handling the SMS is done by batch files as an

automated process. In the server we have scripting language known as awk for formatting the result obtained by running Linux commands in a script file. Secured Shell (SSH) is used for the connection between the server and the monitoring machine, precise for running a command in the server remotely and copying a file from the server.

CHAPTER FOUR: DICUSSION AND EVALUATION

The developed system assists administrators in knowing the status of their servers; this is achieved by either the administrator sending an enquiry about a particular service, or the server sending a message automatically on changing the status of a particular service.

4.1 Development Environment

The specification for the environment where the system was developed and tested is given below:

4.1.1 Hardware Specification

i. CPU: Pentium IV

ii. RAM: 1 GB

iii. GSM Phone: Nokia 3310 with SIM card and RS232 data cable

4.1.2 Software Specification

i. Operating System: Windows XP

ii. Programming Language: Java

iii. Scripting Language: batch file and WMI

iv. J2ME Development Kit: Sun Java Wireless Toolkit for CLDC version 2.5.1

v. Messaging Gateway: Gammu

vi. Database: MySQL

vii. Server's Operating System: Ubuntu

4.2 Functional Structure of the System

The users of this system are expected to be network administrators, who are supposed to have little knowledge of programming that is why we make the message content to be command-like. In other words, some predefined strings are used to specify the type of service that the user wants to check.

When a message is sent by the user, the SMSC forwards the message to a mobile phone having the number specified in the message. This mobile phone at the receiver's end is connected to the machine that monitors the server. Once the message is received, it will be stored in a database for further processing. Depending on the content of the message, a batch file reads the content and performs an operation on the server attached to the machine. Figure 4.1 shows the flow of the whole operation. First if the message sent by the user is BatteryStatus, then a batch file for checking the status of the battery or UPS will be called and send the status back to the user, otherwise it will check if the content is ServerStatus, then find the status of the server and send it to the user. If the message is not the first two then it will check if it is NumberOfUsers, if yes then find the number of users on the server and send it to the user, otherwise the message content is checked if it is Users and send the users who are on the server. The condition goes like this as in Figure 4.1. That is, check the content if it is what you compare find the corresponding operation and terminate the process otherwise move to the next condition until when the user send anything other than the specified string, then a message with all the strings/commands will be send back to the user as reminder in case he/she forgets the commands.

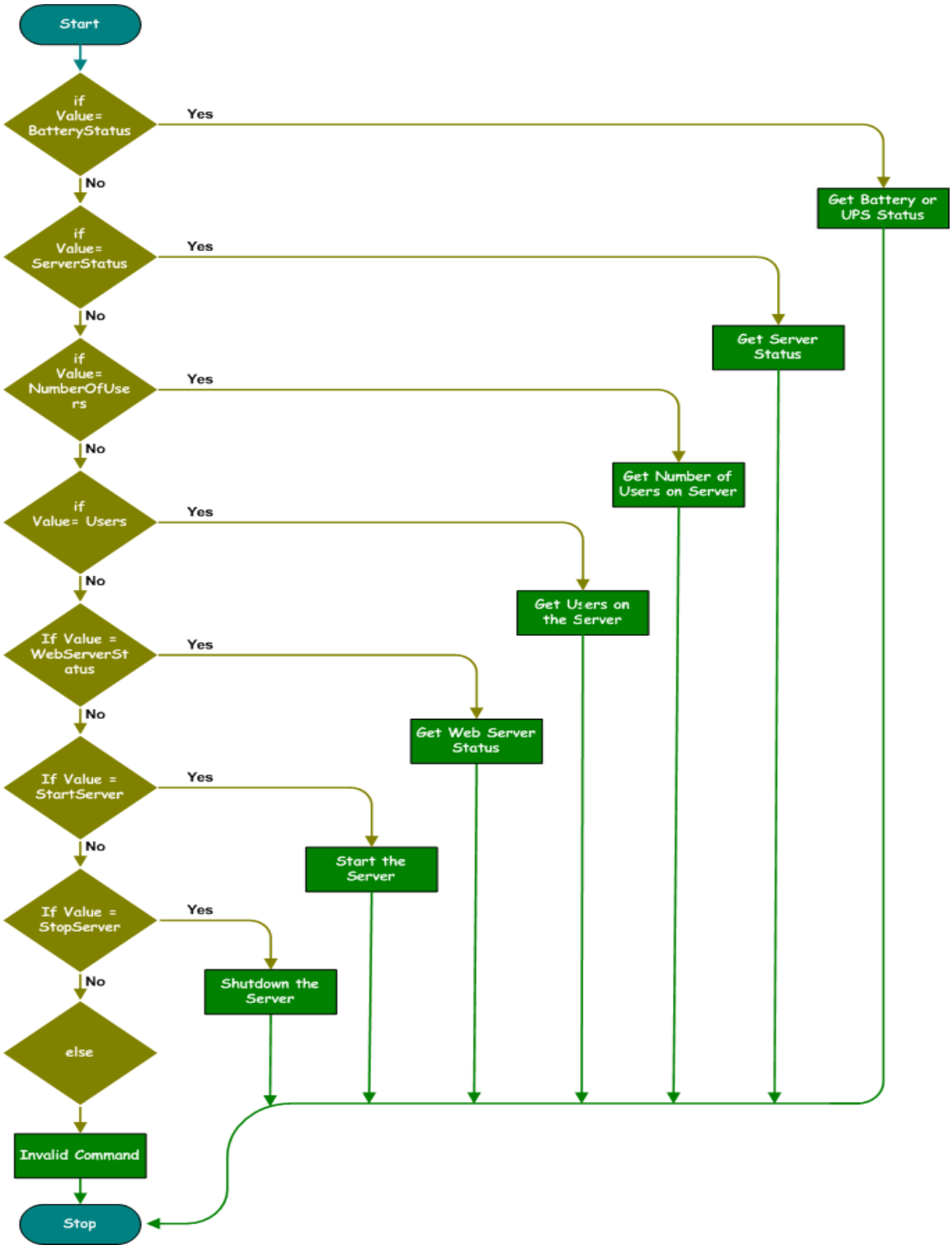


Figure 4.1: System process flow

The main function of this system is to remotely monitor a server via SMS. The first step is to download and install the application on the user's mobile phone. The mobile phone must be Java enabled phone. The files needed for this installation are the JAD and JAR files created during the system development. A confirmation message will appear the moment you start the installation process, asking if the user really wanted to install the application, as depicted in Figure 4.2. Then you follow the installation process.



Figure 4.2: Confirmation message during installation process

The next step is to launch the application. The user must be authenticated before granting access to the application, Figure 4.2 shows the login credentials required to access the application.



Figure 4.2: Login interface

After supplying the right details and the user presses the Login button, then a success message will appear and take the user to where message will be typed as in Figure 4.3, otherwise a login fail message will appear asking the user to try again and take him/her back to login page, as shown in Figure 4.4.



Figure 4.3: Success message when supplying correct login details

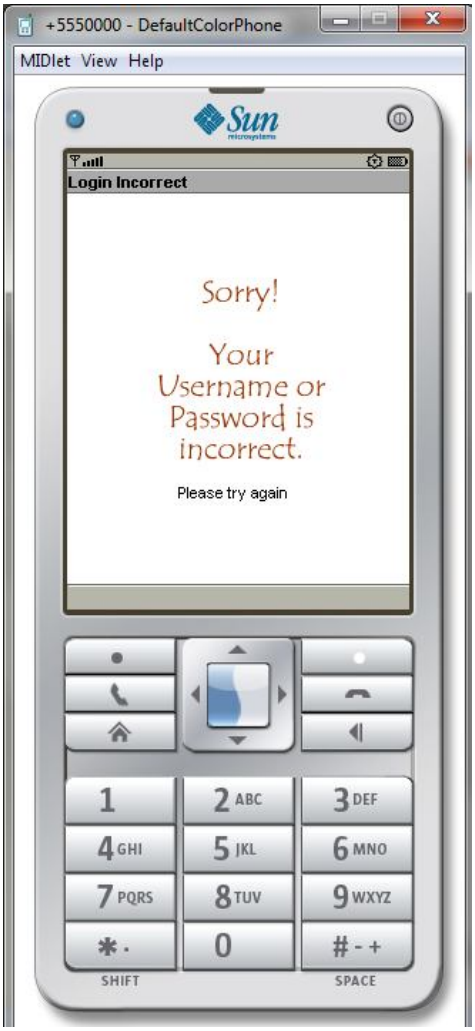


Figure 4.4: Error message when supplying invalid username or password

The user does not need to know the phone number of the mobile phone attached at the backend; it is already configured in to the application and cannot be edited as in Figure 4.5.



Figure 4.5: Sample command typed by the user

The user need to type only the message, in order not to have difficulty in making decision based on the content of the message, the user types a message that looks like a command. The message commands defined in the application that the user is expected to type are shown in Table 4.1. After the user typed the message he/she should press the **Send** button.

Table 4.1: Commands required to be typed by the user

Commands	**Description**
ServerStatus	To check if the server is up or down
BatteryStatus	To know the status of the battery (laptop battery or UPS battery) for both the server and monitoring system
NumberOfUsers	To know how many users are on the server
Users	To know who the users on the server are
WebServerStatus	To know the status of web server
StartServer	used to switch on the server
StopServer	To switch off the server

When the message is received at the backend, it is stored in the database and a batch file called **Main.bat** will start running automatically by one of the property in **smsdrc** known as **RunOnReceive**. This batch file will start a mysql application by reading SQL statements from **Query.sql**. The SQL file has statements for selecting **TextDecoded** (field for keeping the message content) from **inbox** table in the database for the last message received. This record is stored in a text file called **DBaseOutput.txt**. Then another batch file **ExecuteCommand.bat** is called from within **Main.bat**.

ExecuteCommand.bat scans the **DbaseOutput.txt** file for the value stored in it from the previous step, in other words if the content of the message is any of the seven commands above, part of the program in the file **ExecuteCommand.bat** will be executed. Eight cases were looked into as part of the conditional processing. The cases are described below:

Case 1: User sends BatteryStatus.

BatteryStatus is a message sent in order to know the battery status in case of the server being laptop, or Uninterrupted Power Supply (UPS) status for desktop server, the status includes charging/not charging and percentage of the battery. We have two parts here:

i. Battery Status: This is generally for laptop computers, **GetBatteryStatus.bat** is the first batch file called in this module, and it includes WMI script for getting the status of the battery and sends it to a file called **BatteryStatusResult.txt**. Next is a call to another batch file **ReadBatteryVar.bat** which scans the **BatteryStatusResult.txt** for the status and keep it in a global variable, then calls **BatchSqlBattery.bat**. This batch file inserts the value for the variables that represents the status into the database for sending to the user, this is done using ***gammu-smsd-inject*** command.

ii. UPS Status: This part is for desktop server, the operation is like for the Battery Status except that **GetBatteryStatus.bat**, **BatteryStatusResult.txt**, **ReadBatteryVar.bat** and **BatchSqlBattery.bat** are replaced with **GetUPSStatus.bat**, **UPSStatusResult.txt**, **ReadUPSVar.bat** and **BatchSqlUPS.bat** respectively. The functions are almost the same, but for UPS we get the status through a command line tool provided by the manufacturers of the UPS used. The processes involved are shown in Figure 4.6a for laptop server and Figure 4.6b for server being desktop.

In each case when this message is sent, the result will be in respect of both the server and the monitoring system, that is the system will reply with the power status for both systems.

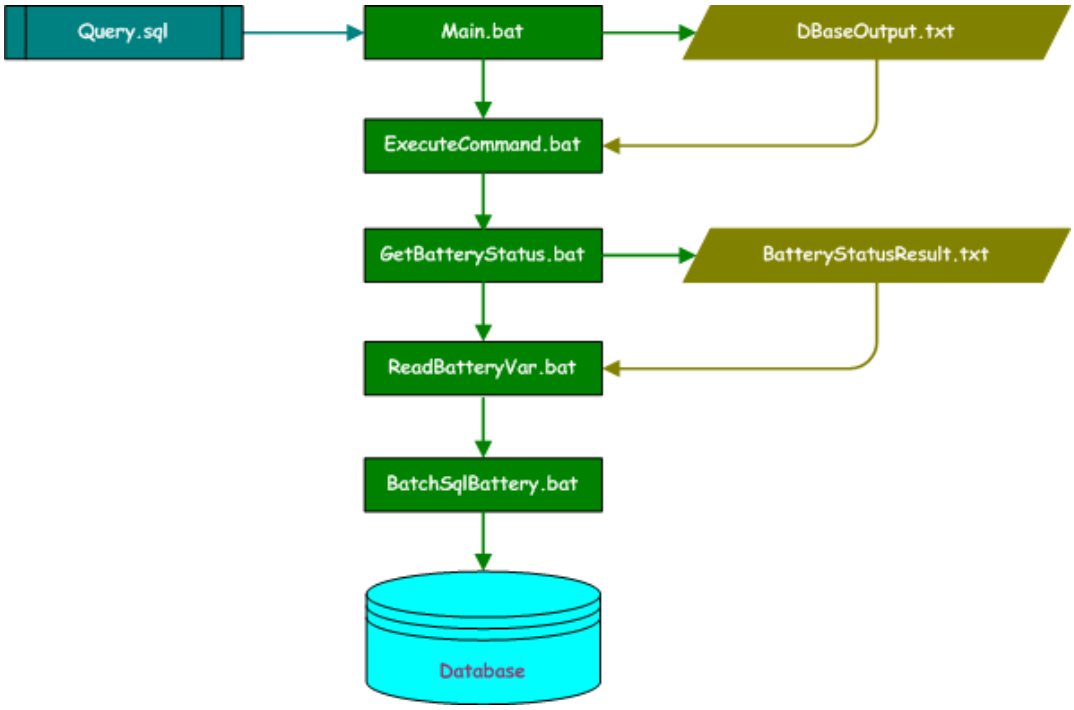


Figure 4.6a: Battery status process flow

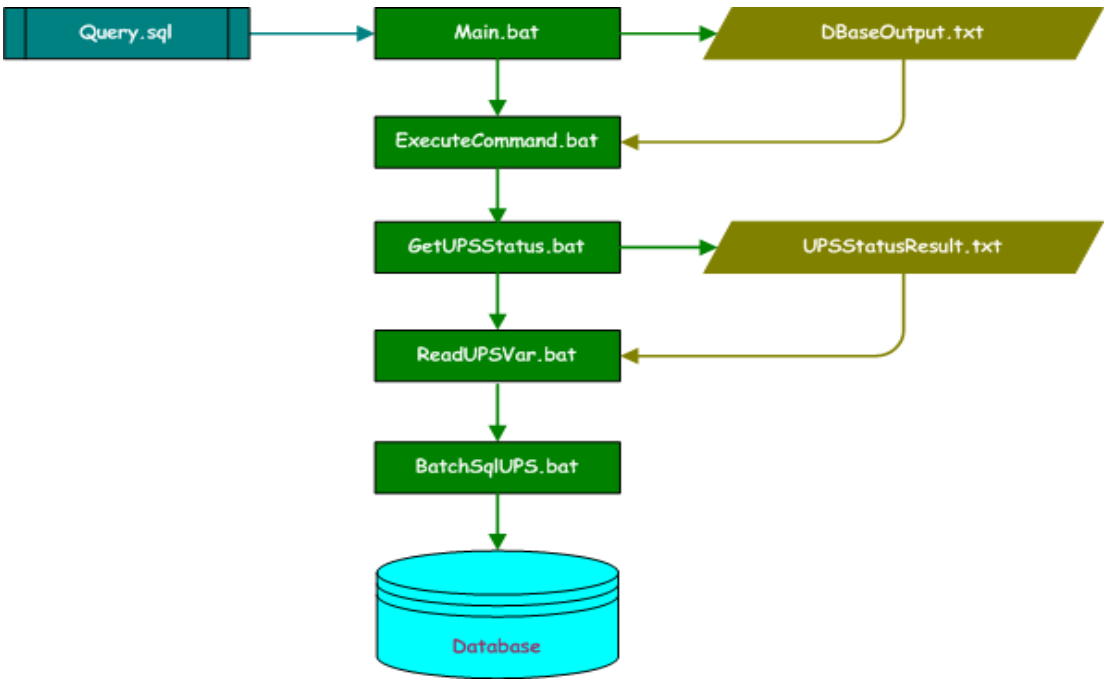


Figure 4.6b: UPS status process flow

Case 2: User sends ServerStatus

ServerStatus is message type that checks if a server is up or down. The first batch file that will be executed is **GetServerStatus.bat** that would ping the IP address of the server and send the result to **ServerStatusResult.txt**. Then a batch called **ReadServerVar.bat** is called, its function is to find some strings in **ServerStatusResult.txt** in order to know the actual status of the server. This status is also sent to **PingResult.txt**. **PingVariables.bat** scans the **PingResult.txt** for the values in it and stores them as global variables, and then a call to **BatchSqlServer.bat** is made to take the global variable's value and insert it into the database.

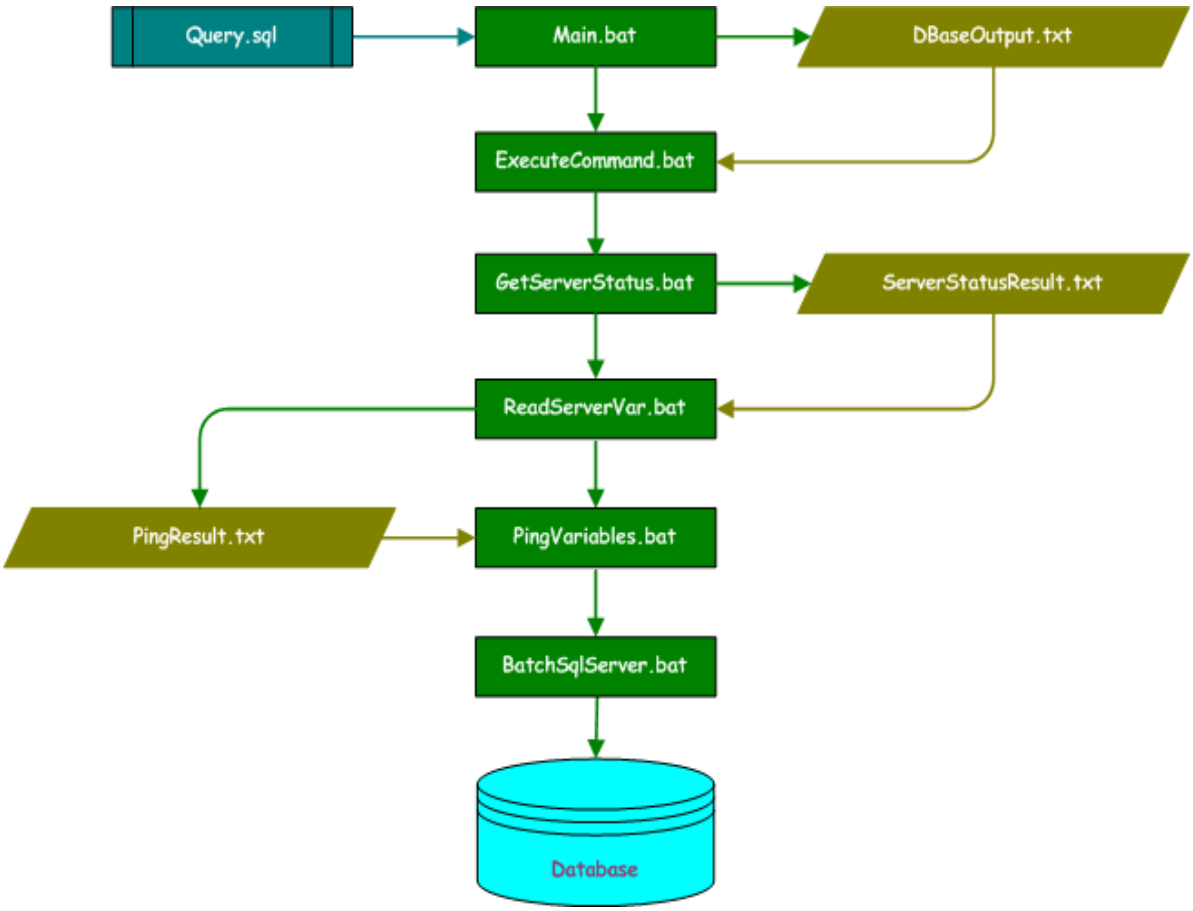


Figure 4.7: Server status process flow

Case 3: User sends NumberOfUsers

Figure 4.8 shows the process of knowing the number of users on the server. **GetNUsers.bat** is the first file executed in this part, the batch file runs a shell script **Users** in the server and sends the result to **NUsers.txt** in the server; then another command for copying the **NUsers.txt** file is executed in the same file to copy the text file from the server to the monitoring machine. Next is a call to **ReadNUsersVar.bat** that gets the number of users and keeps it in a global variable. Finally **BatchSqlNUsers.bat** is also called to insert the global variable value into the database.

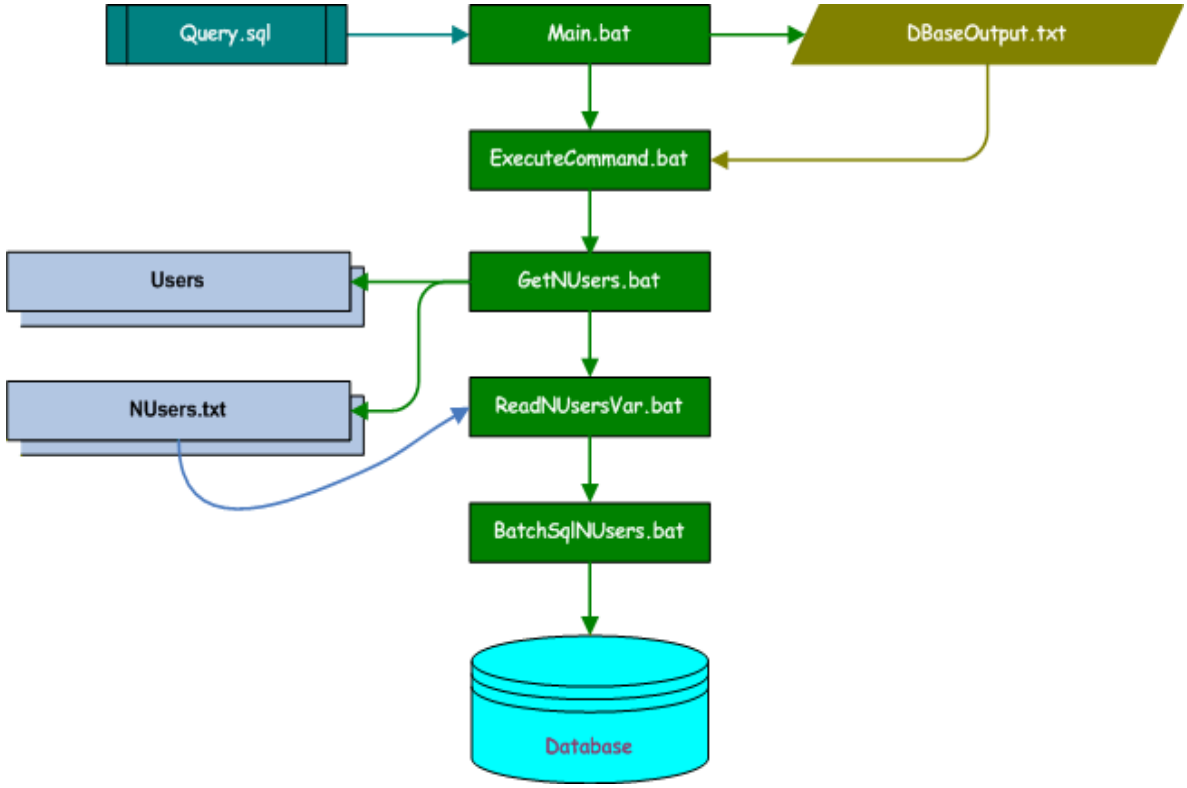


Figure 4.8: Flow diagram for checking number of users.

Case 4: User sends Users

Users is the command that returns the users on the server with their IP address to the user. The first batch file executed when sending this message is **GetUser.bat**, the file consist of two commands one for executing a shell script called **Users** in the server, and the other for copying the file **Users.txt** from the server to the monitoring machine, this file keeps the list of users on the server. However, **ReadUsersVar.bat** is then called to get the users and keeps them in a global variable before making a call to **BatchSqlNUsers.bat**, which will insert the value into the database.

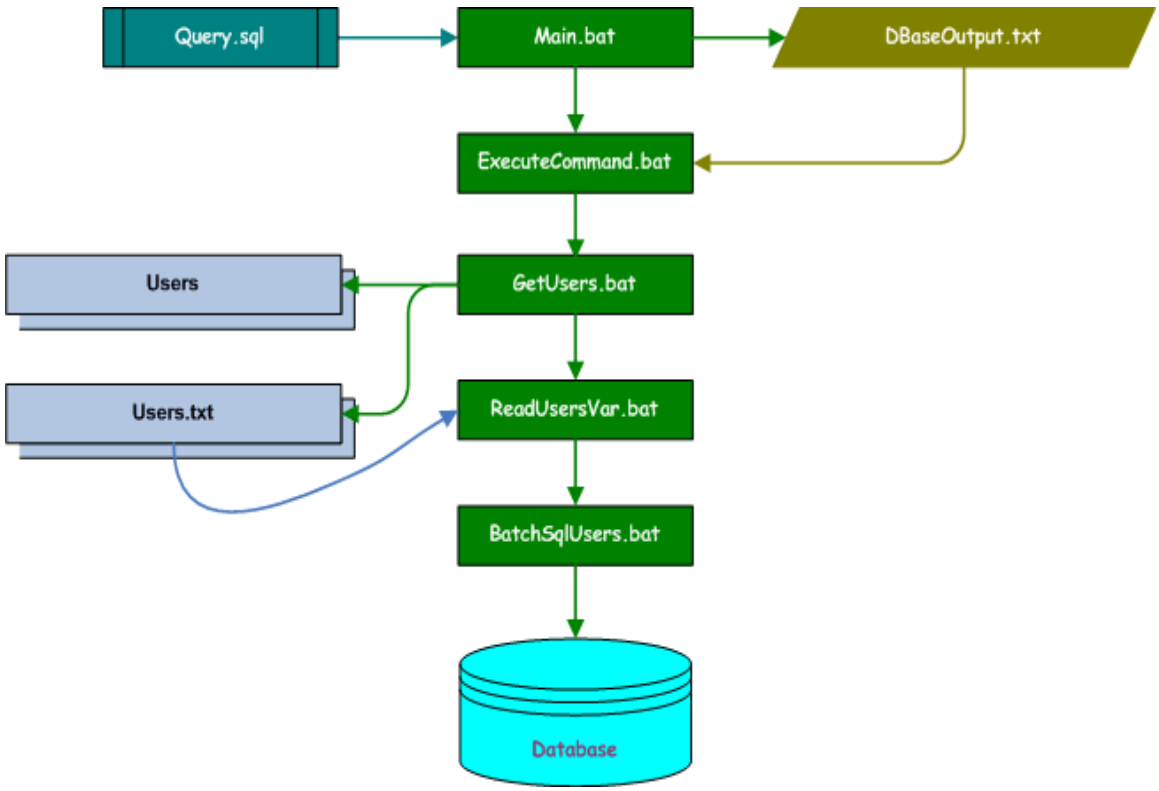


Figure 4.9: Flow diagram for checking the users on the server.

Case 5: User sends WebServerStatus

Knowing the status of the server in case 2 above cannot assure you of the availability of any of the services running on the server, therefore to know the actual status of a service like web server you send a command peculiar to that service. We have a command **WebServerStatus** for knowing the status of web server, when a user sends this command, **GetWebServStatus.bat** will be called to run a script file called **WebServer** in the server and send the result to **WebServStatus.txt**. After copying the file from the server to the monitoring machine then a batch file **ReadWebServVar.bat** is called, in this file we have a decision to make; if the service is running a call to **BatchSqlWebServNR.bat** is made to insert the condition of the service into the database, otherwise **BatchSqlWebServR.bat** is called in case of service not running and insert the result to the database. Figure 4.10 describes the process of checking the status of the web server, by considering the value in **WebServStatus.txt**. In Linux based systems if a service is running, you will see the service name with + prefix after running service command, otherwise the service name will be having – prefix.

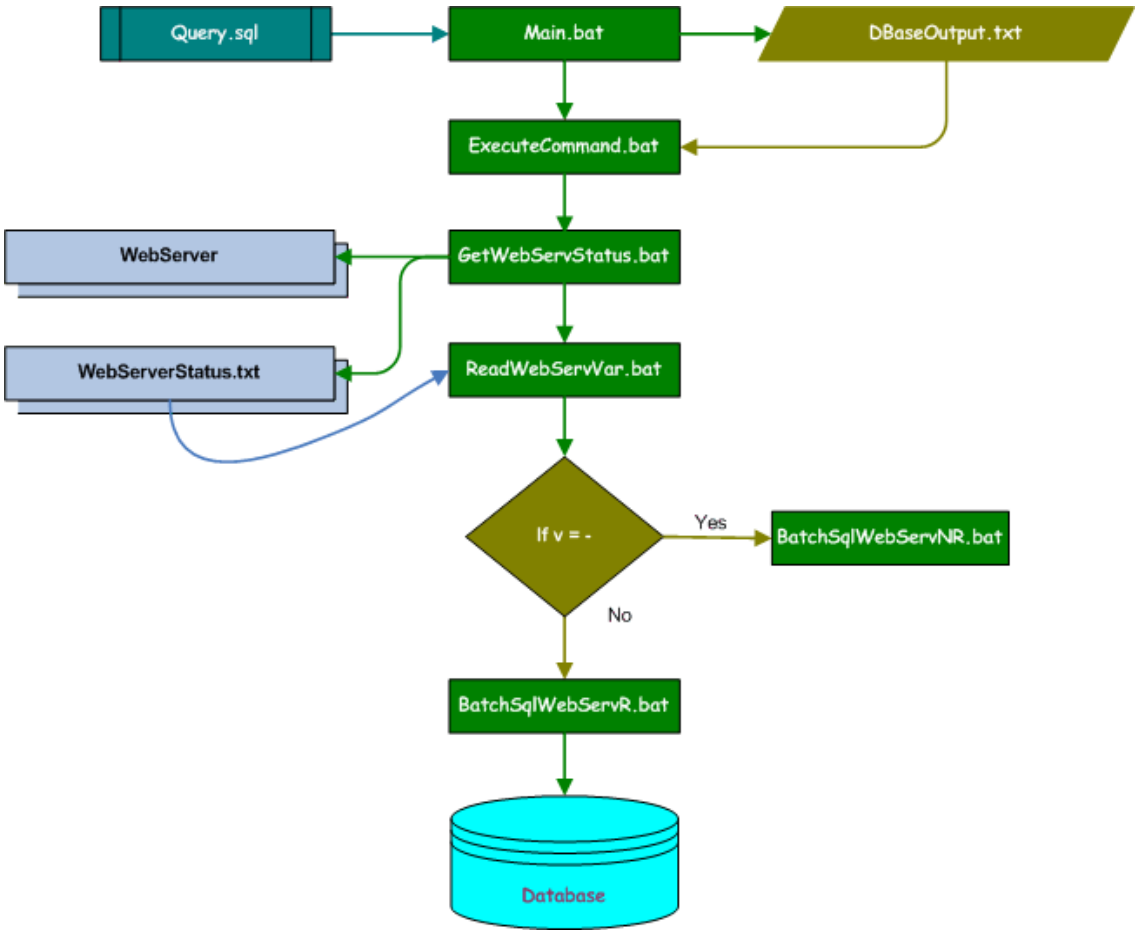


Figure 4.10: Web server status flowchart

Case 6: User sends StartServer

The **StartServer** is used to switch on the server in case of failure. If the server is Desktop then **StartServer.bat** is called which contain statements for setting data pin **D0** to high in order to switch on the server. However, if the server is laptop then a call to **WakeOnLan.bat** is made to wake the computer from shutdown by sending magic packet. In each case **StartReport.bat** is called to send back the report.

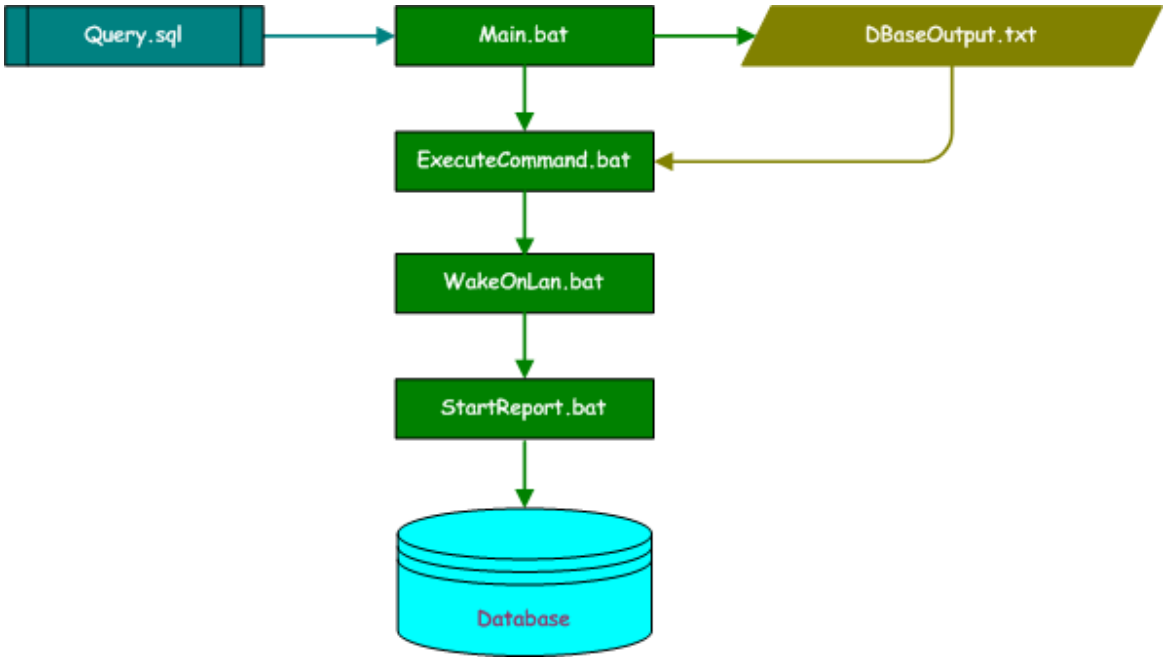


Figure 4.11a: Starting the server by sending magic packet

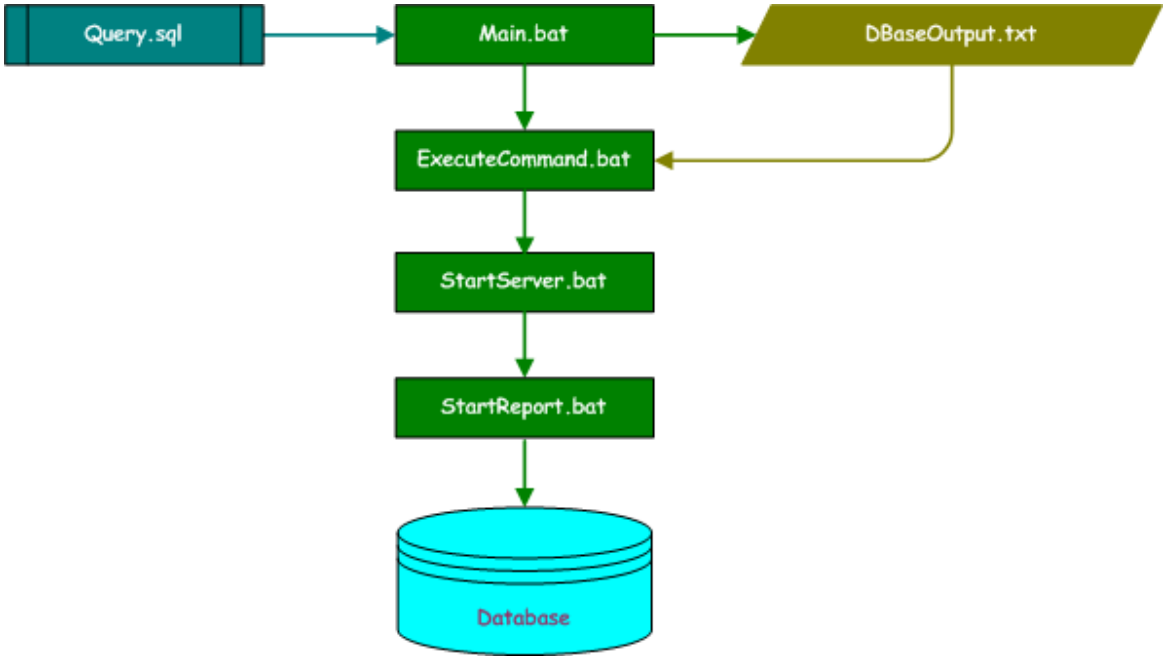


Figure 4.11b: Starting the server by sending data to a circuit.

Case 7: User sends StopServer

If the user wants to stop the server, even though it is unusual but the user can do so by sending the message **StopServer**. For the Desktop server, the batch file **StopServer.bat** is called which contain statements for setting data pin **D1** to high in order to switch off the server. **StopServer.bat** in laptop server contains a statement for shutting down the computer.

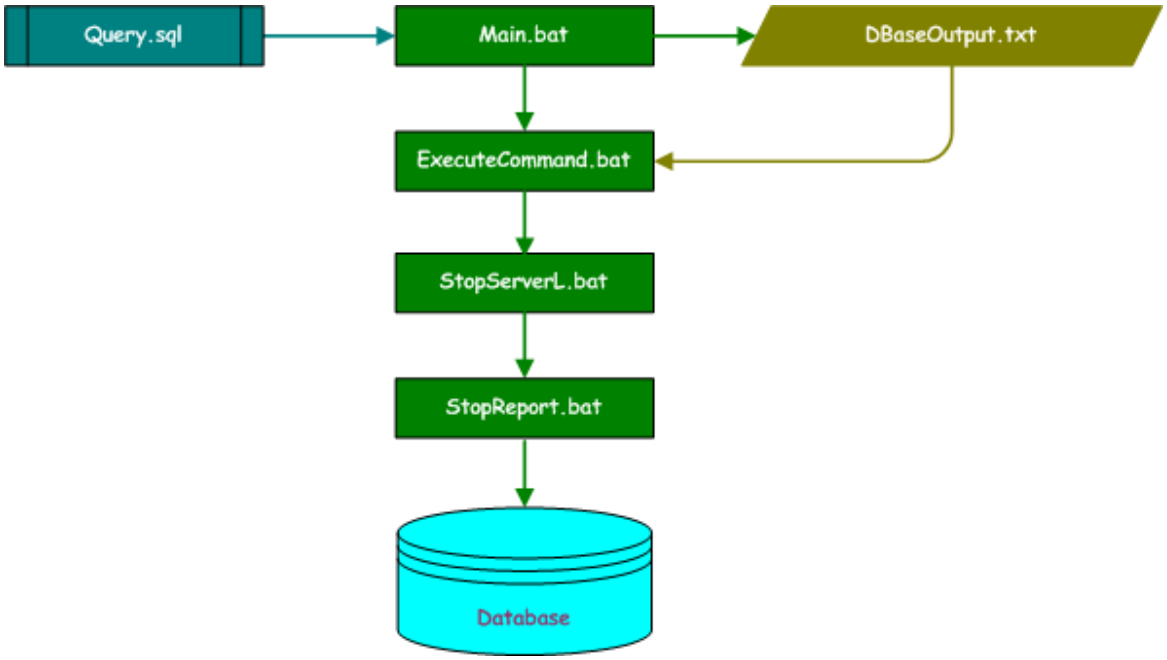


Figure 4.12a: Stopping the server by running a command in the server

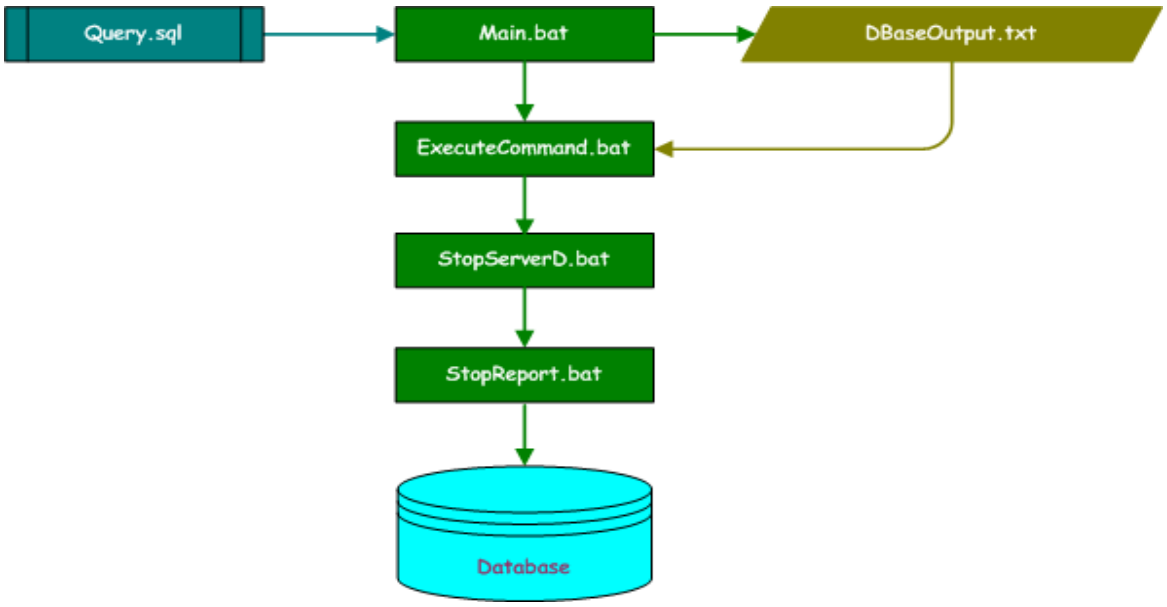


Figure 4.12b: Stopping the server by sending data to a circuit

Case 8: User sends any message

If a user sends any message other than those specified above, **NonCommand.bat** will be called in order to send the user a message containing the commands supported by the system.

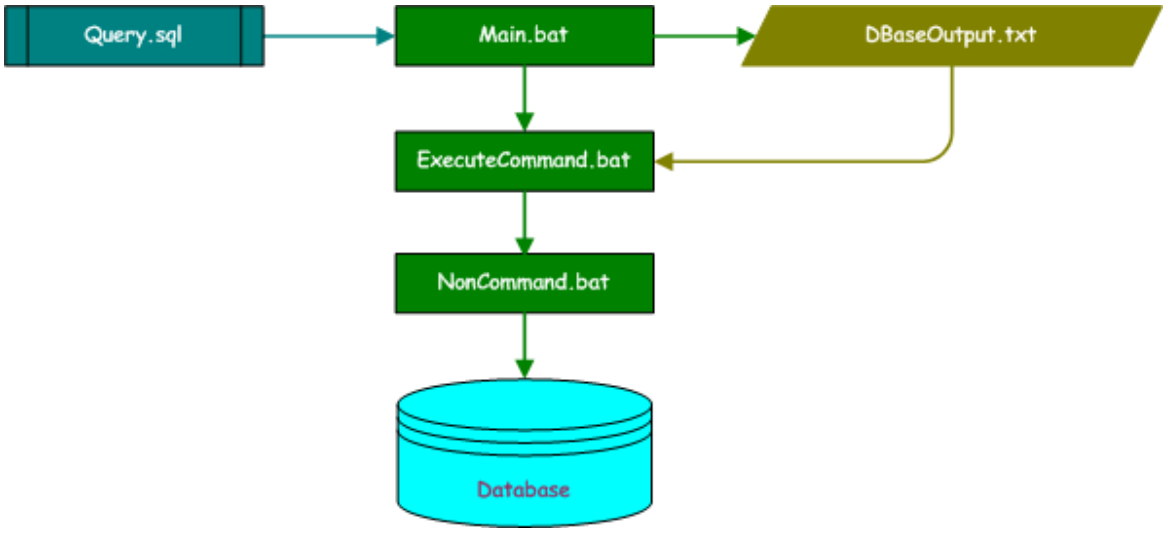


Figure 4.13: Process flow when user sends anything other than the commands specified

There is a batch file called BatteryStatusAlways.bat, that will be set to run regularly (every 10 minutes), this will check the status of the battery and if the battery meter is less than 20%, then a call to a batch file that will shutdown the computer is made. Since the power source for both systems is expected to be the same, so if the server is off due to power outage the user can still send the query for the battery, if the monitoring system is not on charge then there is still power problem, otherwise the power is back and the user can start the system now.

4.3 Performance Evaluation

Even though the time taken for sending a message and getting a reply is a little bit higher than that of running the command on a computer, because it involves some stages which include receiving the message at the backend, processing the message, writing the reply to the database, etc. The manual system involves typing the command and instantly the status is displayed. The proposed system promised increased performance, because if the administrator is away the status of the server can still be known, unlike the manual system that requires the administrator to be physically present.

4.4 Conclusion

This chapter focused on the specification on which the system was tested, and also discussed in more detail the functional components of the system. The process flow for the whole system is shown in this chapter. Each module in the process is taken separately and explained.

CHAPTER FIVE: SUMMARY, CONCLUSION AND RECOMMENDATION

The advancement in computing technology makes mobile phones to become like computers, in the sense that whatever you do with computers you can as well do it with mobile phones. These mobile phones need to be programmed so that they will support the services that computers support; J2ME is one of the popular environments for developing applications that run on mobile phones. This research is on developing a mobile application that is used to monitor and manage server from any location.

5.1 Summary

The objective of this work was to develop a system for remote monitoring and management of servers. A system that achieved this objective was developed and tested. In this system the user sends a message to know the status of the server and/or its services. Also if there is power or service outage a message will be sent to the user as a notification.

The system includes a mobile application designed using J2ME technology for sending messages from user's mobile phones. This application was designed so that only a mobile phone with this application can be used to send the message. Users can of course use any mobile phone even without the application to send the message, but to process a message sent by the user the message must come only from a phone number set in a configuration file called **smsdrc**.

Furthermore, there is a computer that manages the server; this computer has an SMS gateway for receiving a message and stores it in a database. Message handling is done on this computer with the help of batch file programming. The program that handles the messaging is divided into modules; depending on the message sent, a module will be executed. Each module represents

operations like getting the status of the battery, knowing the availability of a service, starting the server, etc.

We have script files for finding the status of some services on the server and for some operations peculiar to the server. We have two different cables that connect the server and the machine that monitors the server. The first connection is Ethernet cable connection used for knowing the status of the server and the services on the server. The other one is from the parallel port of the machine that monitors the server to a circuit and from the circuit to the server's power switch. The latter connection is for turning on and off the server. If the server is a laptop this circuit is not needed because **Wake on LAN** can be used to shutdown and start the server as required.

5.2 Future Work

In order to improve on this research, some areas below ought to be explored.

i. Allow the user to change the username and password.

ii. Include a part in the mobile application that will use GPRS to connect to the server for remote operation.

iii. Make provision for using either SMS or online monitoring.

iv. Reduce the time taken to send message and receive response from the server.

5.3 Conclusion

From the description we provided on our developed system, it is clear that the system achieved all the stated objectives of the research work. We can relate the achievements with the questions we set out to answer in Chapter 1.

A system for monitoring a server remotely was developed, the monitoring is done by sending a message from anywhere to the remote machine for checking the status of the server or services on the server. Also an alert will be sent in form of a message to the user when something happens on the server.

The communication between the server (computer) and a client (mobile phone) which are non compatible devices in terms of sending messages is done in this research using an SMS gateway called **Gammu**. Gammu serves as the interface between the server end and the client end.

A batch file was used to keep checking the server for anomalies. If anything happens, a call to another batch file is made in order to insert the appropriate message into the database to be sent to the user.

For the remote machine to know what to check if a message is sent to it, we make the message content to be a command-like, so that whatever the user sends, it will check if it falls under the commands specified, then it will perform an operation corresponding to that action, otherwise an error message is sent back to the user.

REFERENCES

Alomari, Z. N. M. (2008). UUM Mail Notification System Using Mobile SMS Application. Unpublished M.Sc. Thesis. University Utara Malaysia. Sintok, Malaysia.

AMD (1995). Magic Packet Technology. [Online]. Available: http://support.amd.com/us/Embedded_TechDocs/20213.pdf [Accessed April 19, 2010]

Bezroukov, N. (2010). AWK Programming. [Online]. Available: http://www.softpanorama.org/T-ools/awk.shtml [Accessed September 27, 2010]

Certainty Solutions (2000). *Network Server Monitoring: Where, What, Why and How.* Certainty Solutions Inc., California, USA. [Online]. Available: http://www.certaintysolutions.com/white-papers/monitoring-wp.pdf. [Accessed: May 07, 2010]

Chow, S. (2009). Setting up gammu with smsd. [Online]. Available: http://says-story.blogspot.com/2009/03/setting-up-gammu-with-smsd.html. [Accessed November 06, 2010]

Cihar, M. (2010). Gammu. [Online]. Available: http://wammu.eu/gammu/ [Accessed April 17, 2010]

Computer Hope (2010). Utility Downloads. [Program]. Available: http://www.computerhope.com/dutil.htm [Accessed December 06, 2010]

Cooper, M. (2010). Advanced Bash-Scripting Guide: An in-depth exploration of the art of shell scripting. [Online]. Available: http://tldp.org/LDP/abs/html/ [Accessed March 11, 2010]

Cosenza, F. R. (2005). EMG Enterprise Messaging Gateway Management. Unpublished M.Sc. Thesis. KTH Stockholm-Politecnico Di Torino. Stockholm, Sweden.

Da'na, S. Sagahyroon, A. Elrayes, A. Al-Ali, A. R. and Al-Aydi, R. (2008). Development of a monitoring and control platform for PLC-based applications. *Journal of Computer Standards & Interfaces*, 30:157-166. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0920548907000700. [Accessed October 30, 2010]

Developers' Home(2010). *Short Message Service / SMS Tutorial*. [Online]. Available: http://www.developershome.com/sms. [Accessed: November 20, 2010]

Engdahl, T (2009). Parallel port interfacing made easy: Simple circuits and programs to show how to use PC parallel port output capabilities. [Online]. Available: http://www.epanorama.net/circuits/parallel_output.html. [Accessed December 02, 2010]

Honkasuo, A. Hämäläinen, A. Jäppinen, P. Jääskeläinen, J. and Porras, J. (2006). Mobile Bicycle Monitoring System. pp. 50. Proceedings of Seventh IEEE Workshop on Mobile Computing Systems & Applications held at Semiahmoo Resort, Washington, USA, April 2006.

Hord, J. (2005). *How SMS Works*. How Stuff Works, Inc. [Online]. Available: http://communication.howstuffworks.com/sms.htm. [Accessed: May 15, 2010]

Jaoude, N. B. Tannir, D. and Tannir, M. (2004). *Home Management System using a GPRS-Enabled Mobile Device*. Department of Electrical and Computer Engineering. American University of Beirut, Beirut, Lebanon.

Liaw, J. A. and Ou, S. C. (2003). Design and Implementation of Wireless Web-Service Technology Based on Java. Unpublished M. Sc. Thesis. National Central University Chungli, Chungli, Taiwan, R.O.C.

Lieberman Software (2006) Wake on LAN Technology. [Online]. Available: http://www.liebsoft.com/pdfs/Wake_On_LAN.pdf. [Accessed October 29, 2010]

Litoriya, R. (2009). *SMS Based Home Control System: An Inexpensive Approach*. pp. 345-348. Proceedings of International Conference on Computing Technologies held at Mepco Schlenk Engineering College, Sivakasi, India, December 2009.

Liu, D. (2008). Mobile Phone based Remote Monitoring System. Unpublished M.Sc. Thesis. Auckland University of Technology, Auckland, New Zealand.

Lynch, W. (2006). Introduction to WMIC. [Online]. Available: http://www.net-security.org/dl/articles/WMIC.pdf [Accessed May 26, 2010]

Maxim Integrated Products (2004). Remote monitoring system sends email via SMS. Application Note 3297. [Online]. Available: http://pdfserv.maxim-ic.com/en/an/AN3297.pdf . [Accessed October 30, 2010]

MCTEL (2010). *VAS Development Platform and Turnkey Applications*. [Online]. Available: http://www.mctel.net/art.php/en/ar28/message-gateway.html.html. [Accessed November 01, 2010]

Now SMS (2010). Home of the Now SMS/MMS Gateway. [Online]. Available: http://www.nowsms.com/index.htm. now.sms. [Accessed November 06, 2010]

Oracle Corporation (2011). JSR 178: Mobile Game API. [Online]. Available: http://www.jcp.org/en/jsr/detail?id=178 [Accessed January 17, 2011]

Oracle Sun Developer Network (2005). Java ME: Information Module Profile (IMP) JSR-195. [Online]. Available: http://java.sun.com/products/imp/ [Accessed October, 2010]

Ozeki NG (2010). *SMS Gateway*. [Online]. Available: http://www.ozekisms.com/. [Accessed November 06, 2010]

Sufi, F. Fang, Q. Mahmoud, S. S. and Cosic, I. (2006). *A Mobile Phone Based Intelligent Telemonitoring Platform*. pp. 101-104. Proceedings of the 3rd IEEE-EMBS International Summer School and Symposium on Medical Devices and Biosensors held at MIT, Boston, USA.

Sullivan, M. P. (2004). Secure Remote Network Administration and Power Management. Unpublished M.Sc. Thesis. Naval Postgraduate School Monterey. Monterey, California, USA

Topley, K. (2002). *J2ME in a Nutshell*. O'Reilly & Associates, Inc., Sebastopol, California, United States, pp. 15-31.

Tyson, J. (2000). How Parallel Ports Work. [Online]. Available: http://computer.howstuffworks.com/parallel-port1.htm [Accessed December 02, 2010]

Wake on LAN (2010). Wake-On-LAN Online. [Online]. Available: http://wakeonlan.me/ [Accessed November 06, 2010]

White, J. and Hemphill, D. (2002). *Java 2 Micro Edition: Java in Small Things*. Manning Publication Co., Greenwich, England, pp. 3-33.

Wilansky, E. (2002). WMIC - Take Command-line Control over WMI. [Online]. Available: http://technet.microsoft.com/en-us/library/bb742610.aspx [Accessed May 26, 2010]

Woude R. V. D. (2009). WMI Scripting Examples. [Program]. Available: http://www.robvanderwoude.com/wmiexamples.php [Accessed August 12, 2010]

Wu, C. -H. Jan, R. -H. (2003). System integration of WAP and SMS for home network system. *Journal of Computer Networks*, 42:493-502. [Online]. Available: http://portal.acm.org/citation.cfm?id=859940.859945. [Accessed October 30, 2010]

Xi, H. (2000). Java-Based Intelligent Network Monitoring. Unpublished M.Sc. Thesis. University of Maryland, College Park, Maryland, United States.

ZONITH (2007). *ZONITH SMS Gateway: Server Users Guide Version 4.1*. [Online]. Available: http://www.draware.dk/fileadmin/NMSview/ModemSetup.pdf. [Accessed November 06, 2010]

APPENDIX A: SETTING UP GAMMU ON WINDOWS MACHINE

If you intend using a database for storing the messages, you need to setup the database (e.g. MySQL) before installing Gammu, after which you follow the steps below.

Download Windows Binary file from gammu website

1. Double click the setup and follow the installation instruction. Selecting the installation directory, components to install and adding Gammu to the system path for a particular user or all users are all done during this stage.

2. After you finish installing the Gammu, you create two configuration files (gammurc and smsdrc) in C:/Document and Setting/Administrator. The type and content of the file were discussed in chapter three.

3. Connect the modem or mobile phone to the computer with a serial cable

4. To test the installation you type *gammu—identify* in the command prompt, if the installation is successful properties of the connected mobile phone or modem will be displayed as a result of the above command.

5. Copy and run smsd.sql file form the installation directory to create the MySQL database with the required tables.

6. You can now send SMS from the command prompt or you can develop a program for sending the messages in that case you need to start the daemon (smsd) as a windows service using the command *gammu-smsd –c smsdrc.*

APPENDIX B: CREATING AND RUNNING A PROJECT IN WTK

Installing WTK is a simple process, download the exe file; double click and follow the installation process.

Applications developed in WTK are organized into project, so creating a MIDlet application means creating WTK project. To create projects follow these steps

1. Launch the WTKs' user interface
2. Click New Project. A new window will pop up to enter the project name and MIDlet class name, and then click Create Project.
3. A window for selecting things like, API (Profiles and Configuration), optional packages, MIDlets if more than one, external APIs, etc is displayed; you can leave the default and click OK.
4. In the WTK console (where you clicked Create Project) a message appears telling you that the project has been created, and also showing you where to put the source, resource and library files. By default for any project the source, resource and library file are located in C:\WTK2.5.1\apps\Project_Name\src, C:\WTK2.5.1\apps\Project_Name\res, and C:\WTK2.5.1\apps\Project_Name\lib directories respectively, if WTK is install in C:\.

To run a project you follow the following steps

1. Create Java source file(s) and the optional resource and lib files for the project, place them in the directory specified above.

2. Launch the WTK user interface and click Open Project, a window appears with a list of all the projects created in WTK together with sample projects which are all by default located in C:\WTK2.5.1\apps\, then select the project you want run.

3. Click Build button, to compiles and preverify the source file(s). if there is an error it will be displayed in the console, then you go back to the source file and correct the error otherwise the console displays Build Complete message

4. Click Run button, the emulator will be started and displays all the MIDlets in the project if more than one, then the user select the desired one to launch.

APPENDIX C: SOURCE CODE

gammurc

```
[gammu]

port = com1:

connection = fbus

model = 3310
```

smsdrc

```
[gammu]

port = com1:

connection = fbus

model = 3310

[include numbers]

number1 = 08063238793

[smsd]

Service = mysql

PIN = 1234

LogFile = smsdlog1

User = root

Password =

PC = localhost

Database = smsd

debuglevel = 255

PhoneID = Barroon

RunOnReceive = C:\Main.bat
```

Main.bat

```
@ECHO off
::Remote Server Monitoring System By Barroon Isma'eel Ahmad
ECHO Remote Server Monitoring System
mysql -u root smsd < Query.sql > DBaseOutput.txt
CALL ExecuteCommand.bat
```

Query.sql

```
select TextDecoded from inbox order by ID Desc limit 1
```

ExecuteCommand.bat

```
@ECHO off
:BEGIN
SETLOCAL
SET value=
FOR /f "tokens=*" %%a IN ('TYPE DBaseOutput.txt 2^>NUL') DO SET value=%%a
ECHO value=%value%
IF %value%==BatteryStatus GOTO A
IF %value%==ServerStatus GOTO B
IF %value%==NumberOfUsers GOTO C
IF %value%==Users GOTO D
IF %value%==WebServerStatus GOTO E
IF %value%==StartServer GOTO F
```

```
IF %value%==StopServer GOTO G

IF NOT %value%==StopServer GOTO H

::IF !%value% ==BatteryStatus! && !%value%==ServerStatus! && !%value%==StartServer!
&& !%value%==StopServer! GOTO E

:A

ECHO Battery Status

CALL GetUPSStatus.bat

CALL ReadUPSVar.bat

GOTO END

:B

ECHO Server Status

CALL GetServerStatus.bat

CALL ReadServerVar.bat

CALL PingVariables.bat

GOTO END

:C

ECHO Number of Users on the Server

CALL GetNUsers.bat

GOTO END

:D

ECHO Users on the Server

CALL GetUsers.bat

GOTO END

:E

ECHO Web Server Status
```

```
CALL GetWebServStatus.bat
GOTO END
:F
ECHO Start Server
CALL StartServer.bat
CALL StartReport.bat
GOTO END
:G
ECHO Stop Server
CALL StopServer.bat
CALL StopReport.bat
GOTO END
:H
IF NOT %value%==BatteryStatus GOTO F
:I
IF NOT %value%==ServerStatus GOTO G
:J
IF NOT %value%==StartServer GOTO H
:K
CALL NonCommand.bat
:END
```

GetBatteryStatus.bat

```
@ECHO OFF
```

```
:: Check for Windows NT 4 or later (though XP or later is required)

IF NOT "%OS%"=="Windows_NT" GOTO Syntax

:: Localize variables

SETLOCAL

:: No command line arguments required

IF NOT "%~1"=="" GOTO Syntax

:: Check if WMIC.EXE is available

WMIC.EXE Alias /?:Brief >NUL 2>&1 || GOTO Syntax

:: Use WMI to retrieve battery status information

FOR /F "tokens=1* delims==" %%A IN ('WMIC /NameSpace:"\\root\WMI" Path BatteryStatus
Get Charging^,Critical^,Discharging /Format:list ^| FIND "=TRUE"') DO ECHO Battery is
%%A

FOR /F "tokens=*  delims="  %%A IN ('WMIC /NameSpace:"\\root\WMI" Path BatteryStatus
Get PowerOnline^,RemainingCapacity  /Format:list ^| FIND "="')     DO SET  Battery.%%A

FOR /F "tokens=*  delims="  %%A IN ('WMIC /NameSpace:"\\root\WMI" Path BatteryRuntime
Get EstimatedRuntime              /Format:list ^| FIND "="')     DO SET  Battery.%%A

FOR /F "tokens=*  delims="  %%A IN ('WMIC /NameSpace:"\\root\WMI" Path
BatteryFullChargedCapacity Get FullChargedCapacity            /Format:list ^| FIND "="')     DO
SET  Battery.%%A

:: Calculate runtime left and capacity

SET /A Battery.EstimatedRuntime  = ( %Battery.EstimatedRuntime% + 30 ) / 60

SET /A Battery.RemainingCapacity = ( %Battery.RemainingCapacity%00 +
%Battery.FullChargedCapacity% / 2 ) / %Battery.FullChargedCapacity%


:: Display results

IF /I "%Battery.PowerOnline%"=="TRUE" (

        ECHO Now working on mains power
```

```
        ECHO Now working on main power > BatteryStatusResult.txt
        ECHO Battery %Battery.RemainingCapacity%%% charged
) ELSE (
        ECHO Battery Discharging > BatteryStatusResult.txt
        ECHO Estimated remaining runtime %Battery.EstimatedRuntime% minutes
        ECHO Remaining capacity %Battery.RemainingCapacity%%%
)
echo %Battery.RemainingCapacity% >> BatteryStatusResult.txt
GOTO:EOF
:Syntax
ECHO.
ECHO BattStat.bat,  Version 1.00 for Windows XP Pro or later
ECHO Display current battery status for the local computer
ECHO.
ECHO Usage:  BATTSTAT
ECHO.
ECHO Note:   Uses WMIC to query the battery status;
ECHO         WMIC is native in Windows XP Professional,
ECHO         Windows Server 2003 and Windows Vista.
ECHO.
ECHO Written by Rob van der Woude
ECHO http://www.robvanderwoude.com
:: End localization
IF "%OS%"=="Windows_NT" ENDLOCAL
```

ReadBatteryVar.bat

```
@ ECHO off
setLocal EnableDelayedExpansion
for /f "tokens=* delims= " %%a in (BatteryStatusResult.txt) do (
set /a n+=1
set v!n!=%%a
)
set v
call BatchSqlBattery.bat
```

BatchSqlBattery.bat

```
@ ECHO off
gammu-smsd-inject -c "C:\Documents and Settings\planet\smsdrc" TEXT 08063238793 -text
"%v1% , %v2% "
```

BatteryStatusAlways.bat

```
@ECHO off
CALL GetBatteryStatus.bat
setLocal EnableDelayedExpansion
for /f "tokens=* delims= " %%a in (BatteryStatusResult.txt) do (
set /a n+=1
set v!n!=%%a
)
set v
```

```
if %v2% LSS 20 GOTO A
GOTO B
: A
CALL BatchSqlBattery.bat
:B
EXIT
```

GetUPSStatus.bat

```
@ECHO OFF
ECHO No UPS Installed > UPSStatusResult.txt
ECHO Please Contact your Administrator >> UPSStatusResult.txt
```

ReadUPSVar.bat

```
@ECHO off
setLocal EnableDelayedExpansion
for /f "tokens=* delims= " %%a in (UPSStatusResult.txt) do (
set /a n+=1
set v!n!=%%a
)
set v
CALL BatchSqlUPS.bat
```

BatchSqlUPS.bat

```
@ ECHO off
```

gammu-smsd-inject -c "C:\Documents and Settings\planet\smsdrc" TEXT 08063238793 -text "%v1% , %v2% "

GetServerStatus.bat

```
@ECHO off

ping 127.0.0.1 >ServerStatusResult.txt
```

ReadServerVar.bat

```
@ECHO off

FIND "bytes=" < ServerStatusResult.txt

if not errorlevel 1 GOTO A

FIND "request could not" < ServerStatusResult.txt

if not errorlevel 1 GOTO B

FIND "timed out" < ServerStatusResult.txt

if not errorlevel 1 GOTO C

FIND "Unknown" < ServerStatusResult.txt

if not errorlevel 1 GOTO D

FIND "Destination" < ServerStatusResult.txt

if not errorlevel 1 GOTO E

:A

for /F "tokens=2 delims=()" %%A IN ('find "Rec" ServerStatusResult.txt') DO @ECHO The Server is up with %%A > PingResult.txt

GOTO END

:B

ECHO The server has timed out >PingResult.txt
```

```
GOTO END
:C
ECHO The Server is unknown >PingResult.txt
GOTO END
:D
ECHO The Server is unreachable >PingResult.txt
GOTO END
:E
ECHO The Server is unreachable >PingResult.txt
GOTO END
:END
```

PingVariables.bat

```
@ECHO off
setLocal EnableDelayedExpansion
for /f "tokens=* delims= " %%a in (PingResult.txt) do (
set /a n+=1
set v!n!=%%a
)
set v
call BatchSqlServer.bat
```

BatchSqlServer.bat

```
@ ECHO off
```

gammu-smsd-inject -c "C:\Documents and Settings\planet\smsdrc" TEXT 08063238793 -text "%v1%"

GetServerStatusAlways.bat

```
@ECHO off

ping 192.168.0.1 >ServerStatusResult.txt

CALL ReadServerVarAlways.bat
```

ReadServerVarAlways.bat

```
@ECHO off

FIND "bytes=" < ServerStatusResult.txt

if not errorlevel 1 GOTO A

FIND "request could not" < ServerStatusResult.txt

if not errorlevel 1 GOTO B

FIND "timed out" < ServerStatusResult.txt

if not errorlevel 1 GOTO C

FIND "Unknown" < ServerStatusResult.txt

if not errorlevel 1 GOTO D

FIND "Destination" < ServerStatusResult.txt

if not errorlevel 1 GOTO E

:A

for /F "tokens=2 delims=()" %%A IN ('find "Rec" ServerStatusResult.txt') DO @ECHO The Server is up with %%A > PingResult.txt

GOTO END

:B
```

```
ECHO The server has timed out >PingResult.txt
CALL PingVariables.bat
GOTO END
:C
ECHO The Server is unknown >PingResult.txtCALL PingVariables.bat
CALL PingVariables.bat
GOTO END
:D
ECHO The Server is unreachable >PingResult.txt
CALL PingVariables.bat
GOTO END
:E
ECHO The Server is unreachable >PingResult.txt
CALL PingVariables.bat
GOTO END
:END
```

GetNUsers.bat

```
@ECHO off
plink barroon@192.168.0.1 -pw 123456 ./Users 2> PLinkError.txt
IF %ERRORLEVEL% EQU 0 GOTO A
IF %ERRORLEVEL% GTR 0 GOTO B
:A
ECHO CONNECTED
```

```
pscp -l barroon -pw 123456 192.168.0.1:StatusFiles/NUsers.txt C:\NUsers.txt 2> PSCPError.txt
IF %ERRORLEVEL% EQU 0 GOTO C
IF %ERRORLEVEL% GTR 0 GOTO D
GOTO END
:B
CALL ReadNUsersVar1.bat
GOTO END
:C
CALL ReadNUSersVar.bat
GOTO END
:D
CALL ReadNUsersVar2.bat
GOTO END
:END
```

ReadNUsersVar.bat

```
@ECHO off
setLocal EnableDelayedExpansion
for /f "tokens=* delims= " %%a in (NUsers.txt) do (
set /a n+=1
set v!n!=%%a
)
set v
call BatchSqlNUsers.bat
```

ReadNUsersVar1.bat

```
@ECHO off
setLocal EnableDelayedExpansion
for /f "tokens=* delims= " %%a in (PLinkError.txt) do (
set /a n+=1
set v!n!=%%a
)
set v
call BatchSqlNUsers.bat
```

ReadNUsersVar2.bat

```
@ECHO off
setLocal EnableDelayedExpansion
for /f "tokens=* delims= " %%a in (PSCPError.txt) do (
set /a n+=1
set v!n!=%%a
)
set v
call BatchSqlNUsers.bat
```

BatchSqlNUsers.bat

```
@ ECHO off
gammu-smsd-inject -c "C:\Documents and Settings\planet\smsdrc" TEXT 08063238793 -text
"%v1%"
```

GetUsers.bat

```
@ECHO off
plink barroon@192.168.0.1 -pw 123456 ./Users 2> PLinkError.txt
IF %ERRORLEVEL% EQU 0 GOTO A
IF %ERRORLEVEL% GTR 0 GOTO B
:A
ECHO CONNECTED
pscp -l barroon -pw 123456 192.168.0.1:StatusFiles/Users.txt C:\Users.txt 2> PSCPError.txt
IF %ERRORLEVEL% EQU 0 GOTO C
IF %ERRORLEVEL% GTR 0 GOTO D
GOTO END
:B
CALL ReadUsersVar1.bat
GOTO END
:C
CALL ReadUSersVar.bat
GOTO END
:D
CALL ReadUsersVar2.bat
GOTO END
:END
```

ReadUsersVar.bat

```
@ECHO off
```

```
setLocal EnableDelayedExpansion
for /f "tokens=* delims= " %%a in (Users.txt) do (
set /a n+=1
set v!n!=%%a
)
set v
call BatchSqlUsers.bat
```

ReadUsersVar1.bat

```
@ECHO off
setLocal EnableDelayedExpansion
for /f "tokens=* delims= " %%a in (PLinkError.txt) do (
set /a n+=1
set v!n!=%%a
)
set v
call BatchSqlUsers.bat
```

ReadUsersVar2.bat

```
@ECHO off
setLocal EnableDelayedExpansion
for /f "tokens=* delims= " %%a in (PSCPError.txt) do (
set /a n+=1
set v!n!=%%a
```

)

set v

call BatchSqlUsers.bat

BatchSqlUsers.bat

@ ECHO off

gammu-smsd-inject -c "C:\Documents and Settings\planet\smsdrc" TEXT 08063238793 -text "%v1%,%v2%"

GetWebServStatus.bat

@ECHO off

plink barroon@192.168.0.1 -pw 123456 ./WebServer 2> PLinkError.txt

IF %ERRORLEVEL% EQU 0 GOTO A

IF %ERRORLEVEL% GTR 0 GOTO B

:A

ECHO CONNECTED

pscp -l barroon -pw 123456 192.168.0.1:StatusFiles/WebServerStatus.txt C:\WebServerStatus.txt 2> PSCPError.txt

IF %ERRORLEVEL% EQU 0 GOTO C

IF %ERRORLEVEL% GTR 0 GOTO D

GOTO END

:B

CALL ReadWebServVar1.bat

GOTO END

:C

```
CALL ReadWebServVar.bat
GOTO END
:D
CALL ReadWebServVar2.bat
GOTO END
:END
```

ReadWebServVar.bat

```
@ECHO off
setLocal EnableDelayedExpansion
FOR /f "tokens=1" %%a IN (WebServerStatus.txt) DO SET v=%%a
ECHO %v%
IF %v%==- GOTO A
IF NOT v1==- GOTO B
:A
CALL BatchSqlWebServNR.bat
GOTO END
:B
CALL BatchSqlWebServR.bat
GOTO END
:END
```

ReadWebServVar1.bat

```
@ECHO off
```

```
setLocal EnableDelayedExpansion
for /f "tokens=* delims= " %%a in (PLinkError.txt) do (
set /a n+=1
set v!n!=%%a
)
set v
call BatchSqlWebServer.bat
```

ReadWebServVar2.bat

```
@ECHO off
setLocal EnableDelayedExpansion
for /f "tokens=* delims= " %%a in (PSCPError.txt) do (
set /a n+=1
set v!n!=%%a
)
set v
call BatchSqlWebServer.bat
```

BatchSqlWebServNR.bat

```
@ ECHO off
gammu-smsd-inject -c "C:\Documents and Settings\planet\smsdrc" TEXT 08063238793 -text
"The Server is Down"
```

BatchSqlWebServR.bat

```
@ ECHO off
```

gammu-smsd-inject -c "C:\Documents and Settings\planet\smsdrc" TEXT 08063238793 -text "The Server is Running"

BatchSqlWebServR.bat

@ ECHO off

gammu-smsd-inject -c "C:\Documents and Settings\planet\smsdrc" TEXT 08063238793 -text "%v1%"

StartServer.bat

@ECHO off

lptout 0

lptout 0

sleep 2

lptout 1

lptout 1

sleep 2

lptout 0

lptout 0

StartReport.bat

@ ECHO off

gammu-smsd-inject -c "C:\Documents and Settings\planet\smsdrc" TEXT 08063238793 -text "The Server Started, Successfully"

StopServer.bat

@ECHO off

lptout 0

lptout 0

sleep 2

lptout 2

lptout 2

sleep 5

lptout 0

lptout 0

StopReport.bat

@ ECHO off

gammu-smsd-inject -c "C:\Documents and Settings\planet\smsdrc" TEXT 08063238793 -text "The Server Stopped Successfully"

NonCommand.bat

@ ECHO off

gammu-smsd-inject -c "C:\Documents and Settings\planet\smsdrc" TEXT 08063238793 -text "Invalid Message Type, Use BatteryStatus, ServerStatus, NumberOfUsers, Users, WebServerStatus, StartServer or StopServer"

Users

#A Script to Check Available Users

who | awk '$5 !~ /:0/{print $1,$5}' | sort | uniq > StatusFiles/Users.txt

who | awk '$5 !~ /:0/{print $1,$5}' | sort | uniq | wc –l > StatusFiles/NUsers.txt

WebServer

#A Script to Check if Web Server is Up

service –status-all | awk ‘ /apac/{print $2, $4}’ > StatusFiles/WebServerStatus.txt