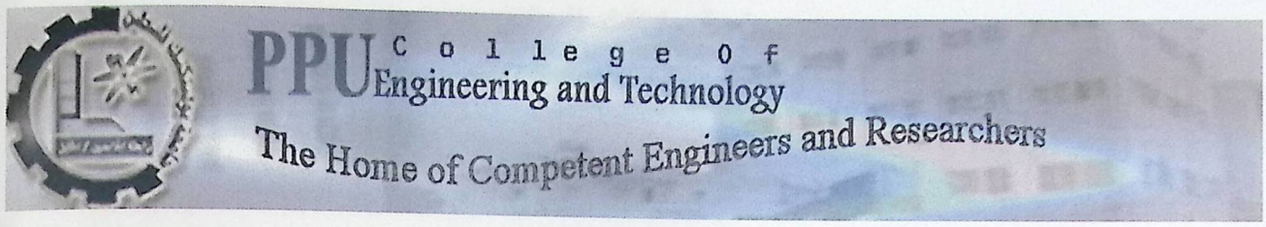


College of Engineering & Technology

Electrical Engineering Department

Communication and Electronics Engineering

Graduation Project

Design and implementation of an SMS to Email Service

Project Team

Asma Tamimi
Salsabeel Rjoub
Heba Fakhori

Project Supervisor
Dr. Murad Abusubaih

Hebron- Palestine
2013

جامعة بوليتكنك فلسطين

الخليل-فلسطين

كلية الهندسة والتكنولوجيا

دائرة الهندسة الكهربائية والحاسوب

اسم المشروع :

Design and Implementation of an SMS to Email Service

أسماء الطلبة :

أسماء التميمي

سلسبيل الرجوب

هبة الفاخوري

بناء على نظام كلية الهندسة والتكنولوجيا وإشراف ومتابعة المشرف المباشر على المشروع وموافقة أعضاء اللجنة الممتحنة تم تقديم هذا المشروع إلى دائرة الهندسة الكهربائية والحاسوب وذلك استكمالا لمتطلبات درجة البكالوريوس في تخصص هندسة الاتصالات والالكترونيات.

توقيع المشرف

د.مراد أبو صبيح .....

توقيع اللجنة الممتحنة

....

توقيع رئيس الدائرة

.....

Acknowledgement

We would like to thank our supervisor Dr. Murad Abusubaih for his guidance and support. We would also like to thank Eng.Sami Salameen and Eng.Omar Abu Saif for their help.

Special thanks also go to our classmates and friends who have given us great suggestions and support while we were working on our project.

Our thanks and appreciations also go to Palestine Information and Communications Technology Incubator (PICTI) for providing us with the financial support that we needed.

Abstract

This project aims to provide two services, which are: SMS to Email Service, File to Email Service using GSM, GPRS and IP Technologies.

These services provide the user many choices in different cases; when there is a need for sending Email while there is no internet connection.

In SMS to Email we use a GPRS modem, which is connected to a server; to receive and send the messages, while a software on the server is used to analyze the data, extract the email address and send the data.

In File to Email Service, we use mobile application on the user side to send the file over the GPRS network to the server; which extracts the data, analyzes it and sends the email.

الملخص

تقوم فكرة هذا المشروع على بناء نظام لتقديم الخدمات التالية للمستخدمين باستخدام تقنيات ال GSM وال GPRS وال IP:

1) خدمة إرسال الرسائل النصية القصيرة(SMS) إلى البريد الإلكتروني.

2) خدمة إرسال الملفات إلى البريد الإلكتروني.

تمنح هذه الخدمات المستخدم عدد كبير من الخيارات في حالات كثيرة عند الحاجة لإرسال إيميل في حالات لا يتوفر فيها اتصال بالانترنت.

في خدمة إرسال الرسائل النصية القصيرة إلى البريد الإلكتروني, GPRS modem موصولة مع ال server سوف تستخدم لإرسال واستقبال الرسائل , بينما يقوم software خاص على ال server باستقبال المعلومات وتحليلها وإرسال الايميل إلى العنوان المطلوب.

في خدمة إرسال الملفات إلى البريد الإلكتروني, سوف يقوم المستخدم بتحميل الملف وإرساله باستخدام تطبيق خاص على جهازه المحمول, ومن ثم سوف ترسل المعلومات عبر شبكة ال GPRS إلى الserver الذي سوف يستقبل المعلومات ويحللها ومن ثم يرسل الايميل إلى العنوان المطلوب .

Table of Contents

List of Figures

List of Abbreviations

n Partnership Project.

ile Phone System.
Research Projects Agency Network.

nter.

el.

roller.
ystem.
Station.

ol Channel.
Multiple Access.
uropean Posts and Telegraphs .
ed Public Data Network.

nced Mobile Phone System.
ol Channel.
ystem.

ates for GSM Evolution.
y Register.

Access Communications System.
mmunication Standards Institute.

sion Multiple Access.

List of Tables

List of Codes

List of Abbreviations

3GPP: Third Generation Partnership Project.

A

AMPS: Advanced Mobile Phone System.
ARPANET : Advanced Research Projects Agency Network.
AT: Attention.
AuC: Authentication Center.

B

BCH: Broadcast Channel.
BS: Base Station.
BSC: Base Station Controller.
BSS : Base Station Subsystem.
BTS: Base Transceiver Station.

C

CCH: Control Channel.
CCCH : Common Control Channel.
CDMA: Code Division Multiple Access.
CEPT : Conference of European Posts and Telegraphs .
CSPDN : Circuit-Switched Public Data Network.

D

D-AMPS: Digital Advanced Mobile Phone System.
DCCH: Dedicated Control Channel.
DCS : Digital Cellular System.

E

EDGE: Enhanced Data rates for GSM Evolution.
EIR : Equipment Identity Register.
Email: Electronic mail
ETACS: Extended Total Access Communications System.
ETSI: European Telecommunication Standards Institute.

F

FDMA: Frequency Division Multiple Access.

G

GGMSC: Gateway MSC.
GSN: Gateway GPRS Support Node
GPRS :General Packet Radio Service
GSM: Global System for mobile communication.

H

HLR: Home Location Register.
HTTP: HyperText Transfer Protocol.

I

IMAP: Internet Mail Access Protocol.
IMEI: International Mobile Equipment Identity.
IMSI: International Mobile Subscriber Identity
IP : Internet Protocol..
ISDN: Integrated Services digital Network.

L

LAI: Local Area Identity.
LTE: Long Term Evolution

M

MDA: Mail Delivery Agent.
ME : Mobile Equipment.
MIME: Multipurpose Internet Mail Extensions.
MMS: Multimedia Messaging Service
MMSC: Multimedia Messaging Service Center.
MO: Mobile Originating.
MS: Mobile Station.
MSC: Mobile Switching Center.
MSISDN: Mobile Station International Subscriber Directory Number
MT: Mobile Terminating.
MTA: Message Transfer Agent.
MUA: Mail User Agent.

N

NMT: Nordic Mobile Telephone.
NSS: Network and Switching Subsystem.

O

OSS: Operating and Support Subsystem.

P

PC: Personal Computer.
PCU: Packet Control
PDA: Personal Digital Assistant
POP: Post Office Protocol.
PSK : Phase Shift Keying
PSPDN : Packet-Switched Public Data Network.
PSTN: Public Switched Telephone Network

R

RBS : Radio Base station.

S

SACCH: Slow Associated Control Channel.
SDCCH: Standalone Dedicated Control Channel.
SGSN: Serving GPRS Support Node.
SIM : Subscriber Identity Module
SM-AL:Short-Message-Application Layer.
SM-LL : Short-Message-Link-Layer..
SM-RL: Short-Message-Relay-Layer.
SMS: Short Message Service
SMSC: Short Message Service Center.
SM-TL: Short-Message-Transfer-Layer.
SMTP: Simple Mail Transfer Protocol.
SPN: Service Provider Name.
SS7: Signaling System No.7.

T

TCH: Traffic Channel.
TDMA: Time Division Multiple Access.
TMSI: Temporary Mobile Subscriber Identity .
TP: Transfer Protocol.
TPDU: Transfer Protocol Data Unit.
TP-MTI: TP-Message-Type-Indicator.
TP-SRI: TP-Status-Report-Indication.
TP-UD: TP-User-Data.

U

URL: Uniform Resource Locator.
USB: Universal Serial Bus.

V

VLR: Visitor Location Register.

W

WAP: Wireless Application Protocol.
WiMax: Worldwide Interoperability for Microwave Access.

Chapter 1

Introduction

1.1 Introduction.
1.2 Motivation.
1.3 Problem Statement.
1.4 Related Work.
1.5 Time Plan.
1.6 Project Estimated Cost.
1.7 Needed Technology.
1.8 Expected Outcomes.
1.9 Report Contents.

1.1 Introduction

SMS (Short Message Service) is one of the most important and common communication services which allow sending and receiving text messages to and from mobile phones. Email (Electronic mail) communication is also one of the preferred means of communication that has the same importance as SMS. It is used widely in many fields. The reason for this is the speed and ease of electronic mail traffic.

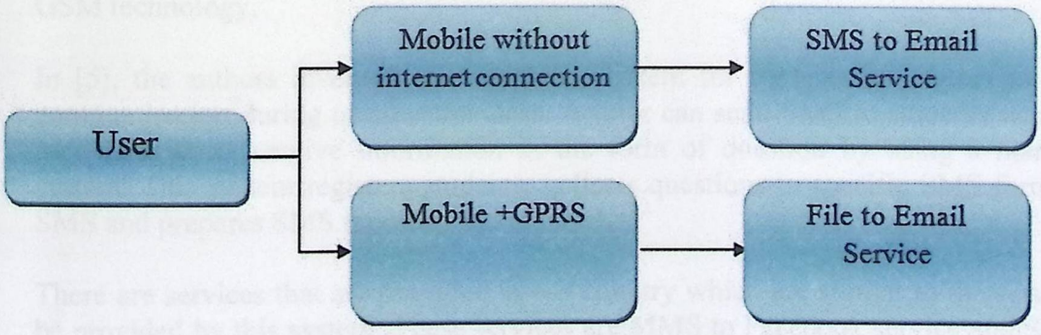
Due to the importance of SMS and Email, this project aims to provide a service called SMS to Email Service. This service allows sending emails from mobile phones through SMS service without an Internet connection. The system also provides additional service which is File to Email service.

SMS to Email service is based on sending messages that contain the email address of the recipient from mobile devices to GPRS modem, which is connected to a PC. Then, a program will convert it to email and send it to the specific email address.

File to Email service is based on sending a file and the email address of the recipient through a mobile application over GPRS network. The file then will be transferred by the radio tower of the mobile operator through the IP network to the server. Then, a program will analyze the data and send it to the specific email address.

1.2 Motivation

The main motivation for providing these services is that they are unavailable in our country. Another motivation is to provide many options for the user in accordance with the situation and capabilities. Fig (1.1) shows the services that are provided to the user.



Fig(1.1): System Services

1.3 Problem Statement

Sometimes you need to send an email in the absence of a computer and Internet connection. This happens when you are outside the home or office or simply at a place where Internet connection is unavailable and all what you have is just your mobile phone. Sometimes you need to communicate with someone that you don't know his/her mobile phone number, but you know his/her email address. However, by SMS to Email service it will be possible for customers to send emails, especially in urgent cases without an Internet connection.

In other cases, the user needs to send files from his/her mobile phone which supports GPRS to email. This will be possible through file to Email service.

1.4 Related Work

SMS (Short Message Service) has achieved a huge improvement in the communication world. "Billions of SMS messages are sent every day. A lot of innovative applications are now built on top of the SMS technology and more are being developed" [1]. There are many projects at the present time about sending SMS in several different technologies; such as GSM modem or SMS gateway, etc. Here we will introduce some of these projects:

In [2], the author has developed and designed a system for sending short messages via a GSM modem to provide security for homes. GSM modem was used to receive SMS from user's mobile phone which enables the controller to take the suitable action such as switching ON or OFF the light, air condition, fan and other devices. Assembly language has been used to integrate the system with the microcontroller and GSM modem interface.

In [3], the author developed a new model in which SMS technology was used to maintain business continuity using SMS Gateway and programmed application packages.

In [4], the authors provided a system for controlling home appliances remotely via SMS using GSM technology.

In [5], the authors invented an interactive system for students and tutors to provide easy communication during or after the class. A tutor can send SMS to students while in the pull model, students receive information in the form of question by using a menu interactive system. This system registers students, collects questions in specific SMS format, classifies SMS and prepares SMS report by using GSM.

There are services that are provided in our country which are similar to the services that will be provided by this system. These services are MMS to Facebook service and SMS to twitter service.

On the other hand, some operators around the world provide these services. However, they do not reveal enough information about the detailed structure of their systems.

1.5 Time Plan

Table (1.1): Time Plan

weeks / Tasks	1,2	3,4	5,6	7,8	9,10	11,12	13,14	15,16	17,18	19,20	21,22	23,24	25,26	27,28	29,30
Task1															
Task2															
Task3															
Task4															
Task5															
Task6															
Task7															
Task8															

Task1: Choosing the appropriate idea.
Task2: Gathering information about the project for a complete understanding of the idea.
Task3: Data Analysis.
Task4: Specification of the needed skills.
Task5: Specification of the project requirements (Hardware and Software).
Task6: Implementation.
Task7: Testing and evaluation.
Task8: Documentation.

1.6 Project Estimated Cost

Table(1.2): Estimated Cost

Component	Units	Estimated Cost($)
Mobile Phone	1	300$
GSM/GPRS modem + server	1	1000$
Human Resources	3	900$
Consultancy	3	200$
Total		2400$

1.7 Needed Technology

1.7.1 GSM Technology

"GSM technology is an open, digital cellular technology used for transmitting mobile voice and data"[6]. It is the most used wireless technology in the world."GSM is a circuit switched system that divides each 200 kHz channel into eight 25 kHz time slots."[6].It operates at either 900MHz or 1800MHz and has a data rate of 64Kbps.It has the advantage of using SIM card.

1.7.2 GPRS Modem

GPRS modem is a hardware device that is used for sending and receiving text messages by using GSM and GPRS technologies. It connects a PC to mobile terminal via cable connected to a COM port, or Infrared and Bluetooth, or Virtual COM port which is created by USB device. By inserting SIM cards in it and by using the appropriate software we will be able to send and receive text messages.

1.8 Expected Outcomes

After the implementation phase, it is expected that the process of sending an SMS from a mobile phone to email would be successful without the need of an internet connection. It is also expected that sending files from mobile phones to email would be also successful. In the case of any error, an automatic notification will be sent to the mobile of the user informing him of error nature. The ultimate purpose of the project is to provide services that are relatively cheap and useful to the people.

1.9 Report Contents

The report consists of seven chapters. Here is a brief description of the topics that are covered in each chapter:

Chapter 1 illustrates the general idea of the project, motivation, problem statement, related work, time plan, estimated cost, needed technologies, and expected outcomes.

Chapter 2 contains details of all the needed technologies and materials.

Chapter 3 contains the design concepts and block diagrams.

Chapter 4 contains all the hardware and software details.

Chapter 5 contains hardware configuration and codes description.

Chapter 6 contains the results, success and failure rates.

Chapter 7 contains the conclusion and the future work.

Chapter 2

Background

2.1 Introduction.
2.2 Generations of Mobile Communications Networks.
2.3 Global System for Mobile Communication (GSM).
2.4 General Packet Radio Service(GPRS)
2.5 Short Message Service (SMS).
2.6 GSM Modem.
2.7 Electronic mail.

2.1 Introduction

This chapter contains details of all used technologies and materials such as GSM technology, SMS, GSM modem and Email.

2.2 Generations of Mobile Communications Networks

2.2.1 First Generation (1G)

The first generation of mobile communications was introduced in the 1980. It was analog, circuit switched, and carried only voice traffic. In 1G, the voice signal was modulated to a higher frequency, typically to 150MHz and up." The first commercially available cellular network using 1G standard was introduced by NTT (Nippon Telegraph and Telephone) in 1979 in Japan"[7]. Analog systems include AMPS, NMT and ETACS. 1G speed varies between 28 Kbps and 56 Kbps.

2.2.2 Second Generation (2G)

The second generation of mobile communications was commercially launched for the GSM standard in Finland by Radiolinja , currently known as Elisa Oyj, in 1991. Voice communications were digitally encrypted which provide higher security and efficient data transfer. 2G networks include GSM, D-AMPS (TDMA) and CDMA. 2G networks also support SMS applications.

GPRS, CDMA2000 are known as 2.5G. GPRS provide data transfer rates from 56-115kbit/s. Services like WAP access, MMS and other internet services like email and World Wide Web access were introduced by 2.5G.

2.75G introduces the evolution of EDGE which uses 8PSK encoding, providing higher data rates up to 236.8Kbps.

2.2.3 Third Generation (3G)

The third generation, 3G, was introduced by NTT DoCoMo in Japan, in 2001. It uses different radio frequencies from 2G. For this reason, it uses different equipment to achieve higher data rates. Data transfer rates range from 384Kpbs to 2Mbps. Many services were provided like video calls, video conferencing, online conference call, mobile TV, online gaming etc. 3G achieves greater security and privacy.

2.2.4 Forth Generation (4G)

4G system provides mobile ultra-broadband Internet access. It introduces applications such as amended mobile web access, IP telephony, gaming services, high-definition mobile TV, video conferencing and 3D television. The first two commercially available technologies

known as 4G were the WiMAX standard and the LTE standard. Fig (2.1) shows Generations of mobile communications networks.

Fig (2.1): Generations of mobile communications networks[8]

2.3 Global System for Mobile Communication (GSM)

2.3.1 GSM History

During the early 1980s, analog cellular telephone systems were achieving a rapid growth in many countries in Europe. Each country developed its own system which differs from other's system in operation and equipment. This led to impracticable roaming of mobile users among international borders [9]. For solving this problem, in 1982, the Conference of European Posts and Telegraphs (CEPT) formed a study group called the Groupe Spécial Mobile committee (GSM). The main task of the committee was to standardize a pan-European cellular public communication network in the 900 MHz radio band[10]. In 1989, GSM responsibility was transferred to the European Telecommunication Standards Institute (ETSI), for maintenance and evolution of GSM specifications. In 2000, the first commercial GPRS services were launched. In March 2004, the GSM association2 reported a total number of 1046.8 million subscribers distributed over 207 countries. Table (2.1) shows the important events of GSM history.

Table (2.1): The important events of GSM history[11]

Years	Events
1982	CEPT establishes a GSM group in order to develop the standards for a pan-European cellular mobile system.
1985	A list of recommendations to be generated by the group is accepted.
1986	Field tests are performed to test the different radio techniques proposed for the air interface.
1987	Time Division Multiple Access (TDMA) is chosen as the access method (with Frequency Division Multiple Access [FDMA]). The initial Memorandum of Understanding (MoU) is signed by telecommunication operators representing 12 countries.
1988	GSM system is validated.
1989	responsibility of the GSM specifications is passed to the European Communications Standards Institute (ETSI).
1990	Phase 1 of the GSM specifications is delivered.
1991	Commercial launch of the GSM service occurs. The DCS1800 specifications are finalized.
1992	The addition of the countries that signed the GSM Memorandum of Understanding takes place. Coverage spreads to larger cities and airports.
1993	Coverage of main roads' GSM services starts outside Europe.
1994	Data transmission capabilities launched. The number of networks rises to 69 in 43 countries by the end of 1994.
1995	Phase 2 of the GSM specifications occurs. Coverage is extended to rural areas.
1996	June: 133 network in 81 countries operational.
1997	July: 200 network in 109 countries operational, around 44 million subscribers worldwide.
1999	Wireless Application Protocol came into existence and 130 countries operational with 260 million subscribers
2000	General Packet Radio Service (GPRS) came into existence.
2001	As of May 2001, over 550 million people were subscribers to mobile telecommunications

2.3.2 GSM Architecture

The GSM system includes four subsystems: Mobile Station (MS), Base Station Subsystem (BSS), Network and Switching Subsystem (NSS), and Operating and Support Subsystem (OSS).

- The MS subsystem involves:

  1. Mobile Equipment (ME) which is the mobile phone itself. Each mobile phone has its unique International Mobile Equipment Identity (IMEI) stored in the ME which identifies uniquely the device when attached to the mobile network.

  2. Subscriber Identity Module (SIM) which is a smart card used to identify subscribers. It contains all subscriber's information, such as; TMSI, IMSI, Service Provider Name (SPN), and Local Area Identity (LAI).

- BSS contains Base Transceiver Station (BTS) and Base Station Controller (BSC). Base Transceiver Station (BTS) is also referred to as the radio base station (RBS). It is a transmission component which handles speech encoding, encryption, multiplexing (TDMA), and modulation/demodulation of the radio signals. BSC is a managing component which manages connections of BTSs. It is responsible for handover, cell site configuration, management of radio resources, and tuning of BTS radio frequency power levels.

- NSS carries out the switching functions of GSM such as call control, charging, mobility management, signaling, subscriber data handling. It contains the following :

  1. Mobile Switching center (MSC): It is responsible of traffic management, call set-up, call routing to a roaming subscriber, termination, charging and accounting information.

  2. Gateway MSC (GMSC): is a gateway used to interconnect the cellular network and the PSTN .It is responsible of routing calls from the fixed network towards a GSM user and it is implemented inside the MSC.

  3. Home Location Register (HLR): is the database of subscriber information such as; subscriber identity (IMSI, MSISDN), subscriber supplementary services, subscriber location, and subscriber authentication information. It stores details of every SIM card issued by the mobile phone operator.

  4. Visitor Location Register (VLR): is a temporary storage location for subscription information for MSs which are within a particular MSC service area.

  5. Authentication Center (AuC): It is used for security purposes by providing parameters for authentication and encryption functions.

6. Equipment Identity Register (EIR): database contains information on the identity of mobile equipment to prevent calls from stolen, unauthorized or defective mobile stations.

- Operation and Support Subsystem: Functional entity which enables the network operator to monitor and control the system. It is also responsible of GSM network maintenance.

The GSM system also communicates with other networks such as the Public Switched Telephone Network (PSTN), Integrated Services digital Network (ISDN), Circuit-Switched Public Data Network (CSPDN), and packet-switched public data network (PSPDN). Fig (2.2) shows the Architecture of the GSM network.

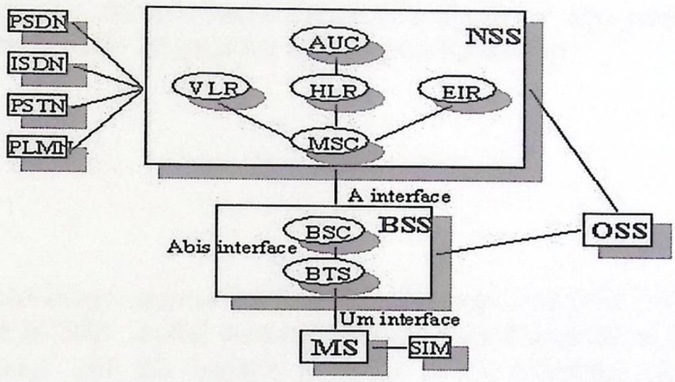


Fig (2.2): Architecture of the GSM network [12]

2.3.3 GSM Services

2.3.3.1 Bearer Services

Bearer services are telecommunication services that provide the capability of transmitting signals between access points and controlling signals between two pieces of equipment. It involves only low layer functions and includes a variety of synchronous and asynchronous data access to PSTN/ISDN and packet switched public data networks. It has a data rate which ranges from 300bps for low speed message transfer to 10Gbps for high speed message transfer.

2.3.3.2 Teleservices

Teleservice uses the capabilities of a Bearer Service to transport data, defining which capabilities are required and how they should be set up. Telesevices are the services offered by a mobile-service network such as; Telephony Teleservice and Emergency Teleservice, and data services like videotext, teletext, SMS, Fax and electronic mail service.

2.3.3.3 Supplementary Services

Supplementary services are provided on top of teleservices or bearer services, and include many features such as caller identification, call holding, call waiting, call forwarding, multi-party conversations, and barring calls from specified numbers or groups.

2.4 General Packet Radio Service (GPRS)

GPRS is one of the first technologies that brought wireless Internet to cell phones. GPRS takes its name because it uses packet switching to deal with data efficiently. Instead of dealing with the communications as a steady stream, the service divides it into small parts of data. This makes GPRS compress more information into a given bandwidth.

2.4.1 GPRS History

2.4.1.1 GPRS Beginning

In 1999, cellular networks began annexation GPRS technology into their infrastructure. The service became available in 2001. Initial data-transmission speed increase to 28 kilobytes per second, but GPRS phones surf the internet at speed of 60 kilobytes per second. Data packaging makes GPRS cost-efficient, because phone users don't pay for steady stream, but they only pay for bursts of data. It also does not consume battery while Web paging or sending text messages.

2.4.1.2 EDGE

Exchanged Data rates for GSM Evolution (EDGE) is a newer technology based on GPRS that has the capability to handle the data much faster. GPRS works for occasional Web browsing; EDGE, works for business with a speed of 473 kilobytes per second. In order to use EDGE in their transmissions, Cellular companies should make more modifications to their equipments.

2.4.1.3 GPRS Today

Although cellular communication keeps improving, upgrading to 3G and 4G networking for instance, GPRS is still in wide use around the world. Mobile phones that don't support 3G use GPRS for texting and email. Smart phones use GPRS when they aren't connecting to a 3G network. GPRS is common enough that one hacker conference in 2011 made hacking into GPRS the focus of its presentations. Most networks that use GPRS do not encrypt it [13].

2.4.2 GPRS Architecture

GPRS is a data network that overlays a second-generation GSM network. This data overlay network provides packet data transport at rates from 9.6 to 171 kbps. Moreover, multiple users can share the same resources simultaneously.

The following figure illustrates GPRS Architecture:

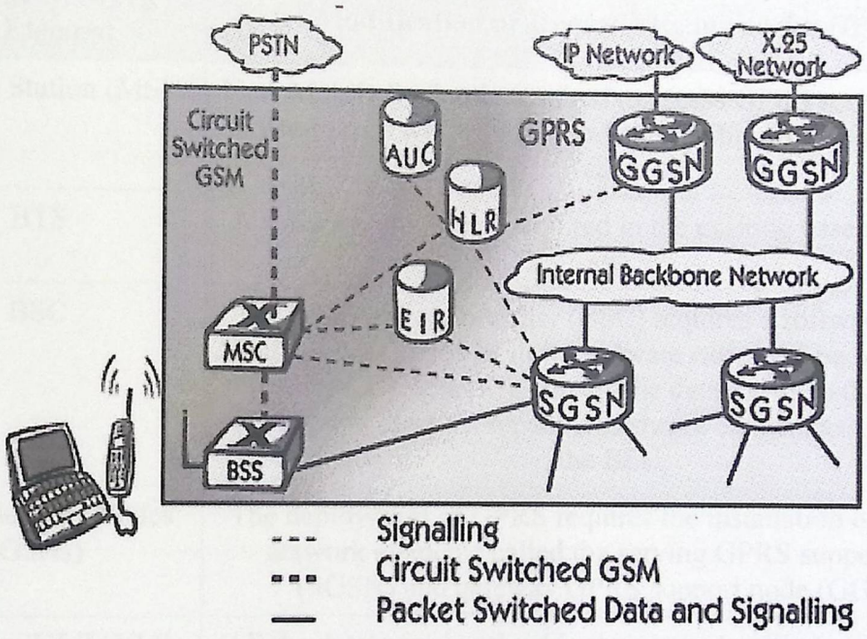


Fig (2.3) : Architecture of the GPRS network

GPRS attempts to reuse the existing GSM network elements as much as possible, some new network elements, interfaces, and protocols for handling packet traffic are required to effectively build a packet-based mobile cellular network. Therefore, GPRS requires modifications to many GSM network elements as summarized below:

Table (2.2) : GSM Network Elements and the corresponding modification required for GPRS

GSM Network Element	Modification or Upgrade Required for GPRS.
Mobile Station (MS)	New Mobile Station is required to access GPRS services. These new terminals will be backward compatible with GSM for voice calls.
BTS	A software upgrade is required in the existing base transceiver site.
BSC	The base station controller (BSC) requires a software upgrade and the installation of new hardware called the packet control unit (PCU). The PCU directs the data traffic to the GPRS network and can be a separate hardware element associated with the BSC.
GPRS Support Nodes (GSNs)	The deployment of GPRS requires the installation of new core network elements called the serving GPRS support node (SGSN) and gateway GPRS support node (GGSN).
Databases (HLR, VLR, etc.)	All the databases involved in the network will require software upgrades to handle the new call models and functions introduced by GPRS.

2.4.2.1 GPRS Mobile Stations

New Mobile Station are required to use GPRS services because existing GSM phones do not handle the enhanced air interface or packet data. A variety of MS can exist, including a high-speed version of current phones to support high-speed data access, a new PDA device with an embedded GSM phone, and PC cards for laptop computers. These mobile stations are backward compatible for making voice calls using GSM.

2.4.2.2 GPRS Base Station Subsystem

Each BSC requires the installation of one or more Packet Control Units (PCUs) and a software upgrade. The PCU provides a physical and logical data interface to the base station subsystem (BSS) for packet data traffic. The BTS can also require a software upgrade but typically does not require hardware enhancements.

2.4.2.3 GPRS Support Nodes

Two new components, called GPRS support nodes (GSNs), are added:

2.4.2.4 Gateway GPRS support node (GGSN)

It is an interface and a router to external networks. The GGSN contains routing information for GPRS mobiles, which is used to tunnel packets through the IP based internal backbone to the correct Serving GPRS Support Node. The GGSN also collects charging information connected to the use of the external data networks and can act as a packet filter for incoming traffic.

2.4.2.5 Serving GPRS support node (SGSN)

The Serving GPRS Support Node is responsible for authentication of GPRS mobiles, registration of mobiles in the network, mobility management, and collecting information for charging for the use of the air interface.

2.4.2.6 Internal Backbone

The internal backbone is an IP based network used to carry packets between different GSNs. Tunneling is used between SGSNs and GGSNs, so the internal backbone does not need any information about domains outside the GPRS network. Signaling from a GSN to a MSC, HLR or EIR is done using SS7.

2.4.2.7 Routing Area

GPRS introduces the concept of a routing area. This is much the same as a Location Area in GSM, except that it will generally contain fewer cells. Because routing areas are smaller than Location Areas, less radio resources are used when a paging message is broadcast.[14]

2.4.3 GPRS Services

GPRS extends the GSM Packet circuit switched data capabilities and makes the following services possible:

- SMS messaging and broadcasting
- "Always on" internet access
- Multimedia messaging service (MMS)
- Push to talk over cellular (PoC)
- Instant messaging and presence—wireless village
- Internet applications for smart devices through wireless application protocol (WAP)
- Point-to-point (P2P) service: inter-networking with the Internet (IP)
- Point-to-Multipoint (P2M) service: point-to-multipoint multicast and point-to-multipoint group calls.

If SMS over GPRS is used, an SMS transmission speed of about 30 SMS messages per minute may be achieved. This is much faster than using the ordinary SMS over GSM, whose SMS transmission speed is about 6 to 10 SMS messages per minute.

2.4.4 Supported Protocols

GPRS supports the following protocols:

- Internet protocol IP.

- Point-to-point protocol (PPP). In this mode PPP is always not supported by the mobile phone operator but if the mobile is used as a modem to the connected computer, PPP is used to tunnel IP to the phone. This allows an IP address to be assigned dynamically (IPCP not DHCP) to the mobile equipment.

- X.25 connections. This is typically used for applications such as wireless payment terminals, although it has been removed from the standard. X.25 can still be supported over PPP, or even over IP, but doing this requires either a network based router to perform encapsulation or intelligence built in to the end-device/terminal; e.g., user equipment (UE).

When TCP/IP is used, each phone can have one or more IP addresses allocated. GPRS will store and forward the IP packets to the phone even during handover. The TCP handles any packet loss [15].

2.5 Short Message Service (SMS)

2.5.1 SMS Definition

Short Message Service is a service of transfer short messages over the mobile networks .It has become the most common service which serves the mobile users in sending and receiving personal text messages, email notification, information services, and so on, from a single user, or several users.

SMS has become the easier service for people to communicate with each other using text messages which can contain at most 160 English characters via mobile devices or Internet connected computers. At this time, bulk SMS can be sent to a large number of people to solve the problem of ads for marketers instead of sending it to several people one after the other.

2.5.2 SMS History

SMS (Short Message Service) messaging in the mobile phones had its beginnings in the mid-1980s. This idea was approved in December 1982. In 1984, the technicians Friedhelm Hillebrand of Germany and Bernard Ghillebaert of France developed the concept of SMS. The goal was to use telephony resources that had been so far unused, at a minimal cost.

In February 1985, the first proposal initiating the development of SMS was given at a GSM meeting in Oslo. SMS was commercially introduced in the telecommunications market in 1992. Until 1999, mobile phone networks did not allow users to send short messages to people subscribed to competing companies.

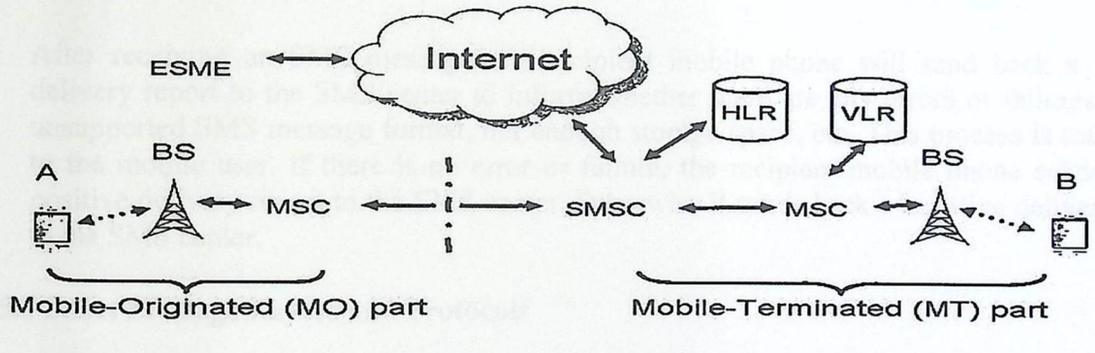
During the first years of its existence, SMS service was rarely used. On the average, only 0.4 messages were sent for every GSM customer each month. In the late 1990s, the service became widely accepted by the mass market. In 2001, an estimated 102.9 billion SMS messages were exchanged worldwide .Gartner Dataquest, one of the industry's major research agencies, expects the number of SMS messages to grow to 146 billion in 2002 and to peak at around 168 billion in 2003 before declining. And by 2006 it had increased to upwards of 205 million[16].

2.5.3 SMS Network Architecture

Elements that can send or receive short messages are named Short Message Entities (SME). An SME can be a software application in a mobile handset but it can also be a facsimile device, telex equipment, remote Internet server, etc. An SME can be a server that interconnects to the SMS centre directly or via a gateway. Such an SME is also known as an External SME (ESME). Typically, an ESME initiate or receive text messages through gateways which bridge the SMS interface to the internet.

The network architecture consists of two segments that are central to the SMS model of operation: the Mobile Originating (MO) part, which includes the mobile handset of the sender, a base station that provides the radio infrastructure for wireless communications, and the originating Mobile Switching Centre (MSC) that routes and switches all traffic into and out of the cellular system on behalf of the sender. The other segment, the Mobile Terminating (MT) part, includes a base station and the terminating MSC for the receiver, also there is a centralized store-and-forward server known as SMS Centre (SMSC) which is responsible for accepting and storing messages, retrieving account status, and forwarding messages to the intended recipients.

It is assisted by two databases: the Home Location Registrar (HLR) which contain permanent mobile subscriber information, and the Visitor Location Registrar (VLR) which contain temporary mobile subscriber information. Figure (2.4) shows the typical network architecture for SMS communication.[17]



Fig(2.4):Typical network architecture for SMS

2.5.4 Basic Concepts of SMS Technology

Here are the basic concepts of SMS technology :[18]

1. Validity Period of an SMS Message

An SMS message is stored temporarily in the SMS center if the recipient mobile phone is offline. After a specified period, the SMS will be deleted from the SMS center so it can not send SMS after that to the recipient mobile phone when it becomes available. This period is called the validity period.

2. Message Status Reports

If the sender wants to know whether an SMS message has reached the recipient mobile phone successfully, he needs to set a flag in the SMS message to notify the SMS center that he wants the status report about the delivery of this SMS message. The status report is sent in the form of an SMS message.

3. Message Submission Reports

After the completion of the SMS, an SMS message goes to the SMS center, then after it reaches SMSC, the SMSC will provide the sender with a message submission report to the mobile phone to inform him if there are any errors or failures such that incorrect SMS message format, busy SMS center, etc. If there is no error or failure, the SMS center sends back a positive submission report to the mobile phone.

4. Message Delivery Reports

After receiving an SMS message, the recipient mobile phone will send back a message delivery report to the SMS center to inform whether there are any errors or failures such as unsupported SMS message format, not enough storage space, etc. This process is transparent to the mobile user. If there is no error or failure, the recipient mobile phone sends back a positive delivery report to the SMS center. Otherwise it sends back a negative delivery report to the SMS center.

2.5.5 Short Message Layers and Protocols

The SMS protocol stack is consist of four layers: the transfer layer, the application layer, the relay layer and the link layer.[19]

2.5.5.1 Transfer layer

SMS-based applications are directly based on the transfer layer. At this layer, the message is considered as a sequence of octets containing information such as message length, message originator or recipient, date of reception, etc. The transfer layer is also known as the SM-TL for Short-Message-Transfer-Layer.

At this layer, the exchange of a message from the originator SME to the recipient SME consists of two to three steps.

Step1: After creation by the originator SME, the message is submitted to the SMSC. The SMSC may verify with other network elements that the message originator is allowed to send messages.

Step2: The SMSC delivers the message to the recipient SME. If the recipient SME is not available for the message delivery, then the SMSC stores the message temporarily until the recipient SME becomes available or until the message validity period expires.

Step3: Upon delivery of the message or upon message deletion by the network, a status report might be transferred back to the originator SME, only if this report had been requested by the originator SME during message submission. The three steps are shown in Fig(2.5)

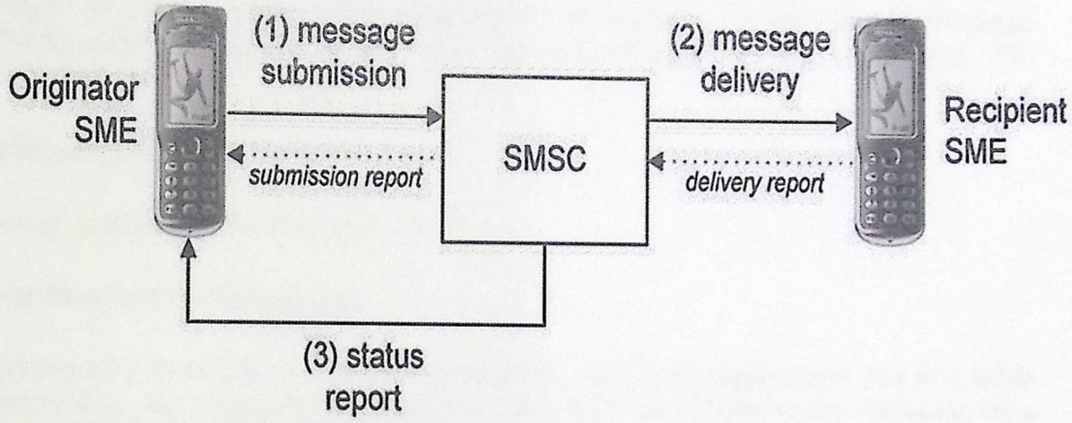


Fig (2.5): Steps of exchange messages on transfer layer

2.5.5.2 Application Layer

The application layer is implemented in SMEs in the form of software applications that send, receive and interpret the content of messages (e.g. message editor, games, etc.). The application layer is also known as SM-AL for Short-Message-Application Layer.

2.5.5.3 Relay Layer

The relay layer allows the transport of a message between various network elements. A network element may temporarily store a message if the next element to which the message is to be forwarded is not available. At the relay layer, the MSC handles two functions in addition to its usual switching capabilities. The first function called SMS gateway MSC (SMS-GMSC) is used for receiving a message from an SMSC and interrogating the HLR to obtain routing information and further to deliver the message to the recipient network. The second function called SMS Inter Working MSC (SMS-IWMSC) which is responsible of receiving a message from a mobile network and submitting it to the serving SMSC. The relay layer is also known as the SM-RL for Short-Message-Relay-Layer.

2.5.5.4 Link Layer

The link layer allows the transmission of the message at the physical level. For this purpose, the message is protected to cope with low-level channel errors. The link layer is also known as the SM-LL for Short-Message-Link-Layer. For transport purposes, an application maps the message content and associated delivery instructions onto a Transfer Protocol Data Unit (TPDU) at the transfer layer. A TPDU is composed of various parameters indicating the type of the message, specifying whether or not a status report is requested, containing the text part of the message, etc. Each parameter is prefixed by the abbreviation TP for Transfer Protocol such as TP-Message-Type-Indicator (TP-MTI), TP-Status-Report-Indication (TP-SRI), TP-User-Data (TP-UD), etc.

2.5.6 Applications of SMS

There are many application for SMS such as: [20]

2.5.6.1 Person-to-Person Text Messaging

Person-to-person text messaging is the most commonly used SMS application and it is what the SMS technology was originally designed for. This use case relates to the exchange of a short text message between two mobile subscribers.

2.5.6.2 Provision of Information

Many content providers make use of SMS text messages to send information such as news, weather report and financial data to their subscribers. Many of these information services are not free. Reverse billing SMS is a common way used by content providers to bill their users. The user is charged a certain fee for each reverse billing SMS message received. The fee will either be included in the monthly mobile phone bill or be deducted from prepaid card credits.

2.5.6.3 Downloading

SMS messages can carry binary data so SMS can be used as the transport medium of wireless downloads. Objects such as ringtones, wallpapers, pictures and operator logos can be encoded in one or more SMS messages depending on the object's size.

2.5.6.4 Alerts and Notifications

SMS is a very suitable technology for delivering alerts and notifications of important events. This is because a mobile phone is a device that is carried by its owner most of the time. Here are some common examples of SMS alert and notification such as:

1. Email, Fax and Voice Message Notifications.
2. Commerce and Credit Card Transaction Alerts.

3. Stock Market Alerts.
4. Remote System Monitoring.

2.5.6.5 Two-way Interactive Text Messaging Applications

SMS messaging technology can be used as the underlying communication medium between wireless devices and servers in a two-way interactive text messaging application.

2.5.6.6 SMS Marketing

SMS messaging can be used as a marketing tool. An example is an SMS newsletter system.

2.6 GSM Modem

2.6.1 GSM Modem

A GSM modem is a wireless modem that works with a GSM wireless network ,it sends and receives data through radio waves.

The GSM net used by cell phones provides a low cost, long range, wireless communication channel for applications that need connectivity rather than high data rates.

A GSM modem is a specialized type of modem that operate over subscription based wireless networks, similar to a mobile phone, it is accepts a SIM card, and it looks just like a mobile phone for a computer.

When a GSM modem is connected to a computer, this allows the computer to use the GSM modem to communicate over the mobile network. While these GSM modems are most frequently sometimes used to provide mobile internet connectivity, many of them can also be used for sending and receiving SMS and MMS messages.[21]

A GSM modem can be an external device or a PC Card / PCMCIA Card. Typically, an external GSM modem is connected to a computer through a serial cable or a USB cable. A GSM modem in the form of a PC Card / PCMCIA Card is designed for use with a laptop computer.[22]

GSM modems convert digital data to Short Message Service (SMS) messages which are small bursts of data for sending and receiving messages over the wireless network.
GSM technology are used in many third-generation (3G) and fourth-generation (4G) smart phones ,these phone can be acting as a GSM modem through connected to the computer via a USB cable or wirelessly over a wide-fidelity (Wi-Fi) network or via Bluetooth. so that they can act as wireless hotspots for computers. This allows users to access the Internet with their smart rather than a traditional Wi-Fi connection. This solution is less expensive than paying

for a separate Internet connection, or for paying connection fees or memberships at public Wi-Fi hotspots.[23]

2.6.2 GPRS Modem

General Packet Radio Service (GPRS) modem is a GSM modem with additional support for GPRS technology for data transmission. It is based on a packet-switched technology, as an extension to GSM. GPRS has a much higher data transmission speed than GSM.

GPRS can be used as the bearer of SMS. This is much faster than SMS over GSM. A GPRS modem is required to send and receive SMS via GPRS. Some wireless carriers do not support the sending and receiving of SMS via GPRS. A GPRS modem is typically required for MMS.[24]

2.6.3 AT Commands

AT commands are instructions used to control a modem. AT is the abbreviation of attention. Every command line starts with "AT" or "at". That's why modem commands are called AT commands. There are two types of AT commands:

1. Basic commands are AT commands that do not start with a "+". For example, D (Dial), A (Answer), H (Hook control), and O (Return to online data state) are the basic commands.

2. Extended commands are AT commands that start with a "+". All GSM AT commands are extended commands. For example, +CMGS (Send SMS message), +CMGL (List SMS messages), and +CMGR (Read SMS messages) are extended commands.[25]

2.7 Electronic mail

2.7.1 Email History

Considering the definition of the email as messages sent electronically, the first appearance of email messages was in the last century with telegraph messages (by wire) and Morse Code transmissions (via airways). Telex network was developed in 1926 and was used extensively by business until 1980. It is a switched network of teleprinters similar to a telephone network, for the purposes of sending text-based messages using standard signaling techniques. In 1950, Fax machines were developed.

During the 1960's and 1970's many companies who were using mainframe and mini computers also used email facilities on those systems. In 1971, the ARPANET ("Advanced Research Projects Agency Network") appeared as the first large network of computers. It was sponsored and created by the U.S. Department of Defense which later led to the development of the internet. During this period, Ray Tomlinson invented internet based email in which he

sent the first email between two adjacent computers using the ARPNET network. He picked the @ symbol to denote sending messages from one computer to another.[26]

By 1974, Email was frequently used for military purposes. By 1976 email had really taken off, and commercial packages began to appear. Within a couple of years, 75% of all ARPANET traffic was email. In 1977, Crocker, John Vittal, Kenneth Pogran, and D. Austin Henderson collaborated on a DARPA initiative to collect various email data formats into a single, coherent specification, resulting in RFC 733. In 1980, the first version of the electronic system was designed and developed for use at University of Medicine and Dentistry of New Jersey (UMDNJ).

In 1997, Microsoft Outlook was released .In 2004, Multimedia emails were introduced. By 2009, iphone 3G and other mobile devices make email more accessible.

2.7.2 Email Protocols

The mail system consists mainly of two protocols SMTP and POP3 to send and receive mail respectively. Others may use IMAP4 to receive mail, especially where bandwidth is limited.

2.7.2.1 Simple Mail Transfer Protocol (SMTP)

SMTP protocol is a TCP/IP protocol used to transfer messages from one mail server to another and to upload messages from UA (mail client) to MTA (mail server). SMTP uses 'push' operation, meaning that the connection is initiated by the sending server rather than the receiver.

2.7.2.2 Post Office Protocol (POP3)

POP Protocol is a standard mail Protocol used to receive emails from a remote server to a local email client. It also provides a simple way for users to access mailboxes and download messages to their computers. It works in ' pull ' mode so the receiving PC is responsible of initiating the connection.

2.7.2.3 Internet Mail Access Protocol (IMAP)

IMAP is a mail protocol used for accessing email on a remote web server from a local client. IMAP supports both online and offline modes of operation. It enables multiple clients to manage the same mail box. Fig (2.6) shows the email protocols (SMTP, POP, and IMAP).

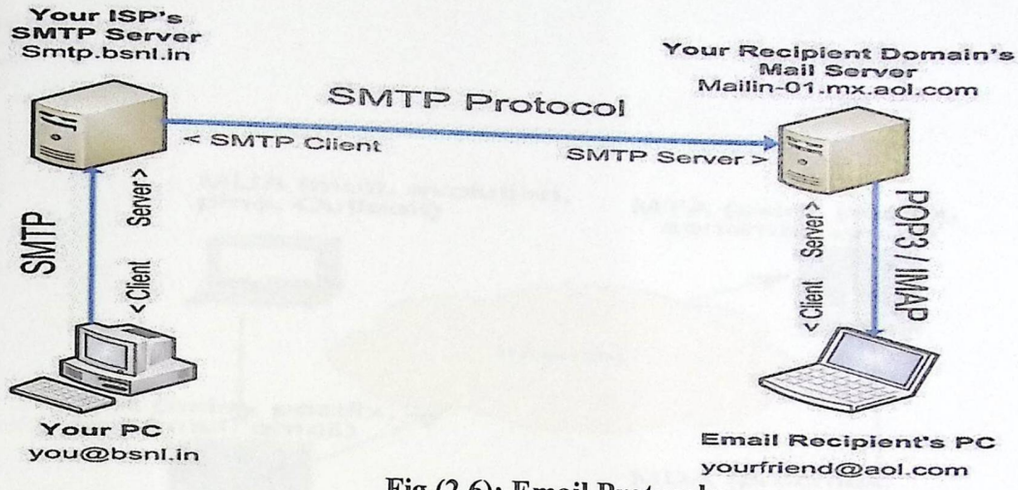


Fig (2.6): Email Protocols

2.7.3 Email Architecture

Email Architecture consists of the following components:

1. Mail User Agent (MUA): an application with which users can create, view, send, and receive emails. It is located on a client system. Such as workstation or PC. Microsoft Outlook Express, Mozilla Thunderbird, and Mutt Email Client are examples of MUA.

2. Mail Transfer Agent (MTA): software that transfers email messages from one computer to another using a client–server application architecture. Postfix, Sendmail, Microsoft Exchange, and Lotus Domino Server are examples of MTU.

3. Mail Delivery Agent(MDA): a computer software component that is responsible for the delivery of e-mail messages to a local recipient's mailbox. Fig(2.7) illustrates the Email Architecture.

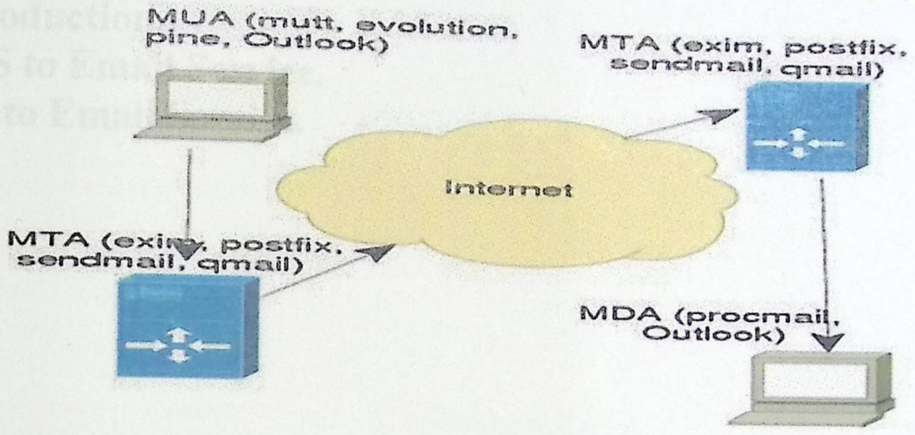


Fig (2.7): Email Architecture

Chapter 3

Conceptual Design

3.1 Introduction.
3.2 SMS to Email Service.
3.3 File to Email Service.

3.1 Introduction

This chapter presents the design concepts of the system and the block diagrams that help to understand how the system works.

3.2 SMS to Email Service

3.2.1 System Block Diagram

The system consists of the following basic components:

1. Mobile Phone.
2. GPRS Modem.
3. Server.

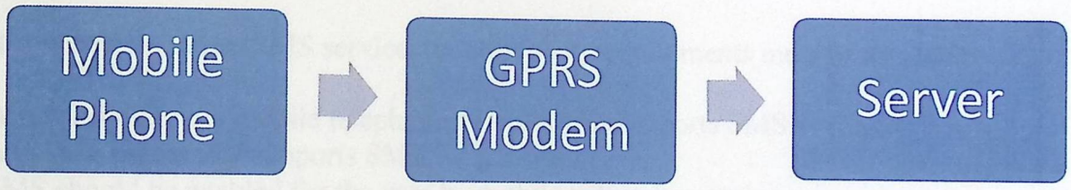


Fig (3.1): Preliminary Design of SMS to Email Service

A detailed figure illustrating the system entities is shown in fig(3.2)

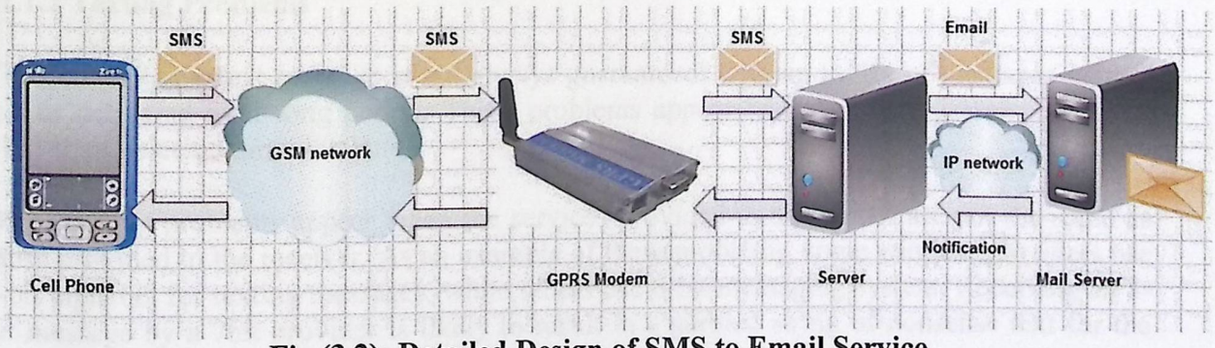


Fig (3.2): Detailed Design of SMS to Email Service

The user sends the message which contains the email address of the recipient through SMS application from his mobile phone to the GPRS modem which is connected to the sever through serial port. The server then will get the message, analyze the data by extracting the message text body and the email address of the recipient and send it to the mail server which forwards it to the specific email address.

3.2.2 System Entities

3.2.2.1 Mobile Phone

The mobile phone should support SMS service. This service enables the users to send messages from mobile phone through the GSM network.

3.2.2.1.1 Short Message Service Application

Most types of mobile phones support the messaging application. The exact procedure of sending SMS differs from device to device, but it usually involves entering the recipient's phone number, and composing the message with the phone keypad. The message instantly can be sent immediately or can be stored for later delivery.

3.2.2.1.1.1 Requirements of using SMS service

In order to be able to use SMS service, the following requirements must be available:

1. A subscription to a mobile telephone network that supports SMS.
2. A mobile phone that supports SMS.
3. SMS should be enabled for the user by mobile network operator.
4. The user must have the knowledge of how to send and receive short messages using his specific model of mobile phone.
5. A destination to send the short message to, or receive message from which is usually another mobile phone.

3.2.2.1.1.2 Texting Problems

The delivery of a text message is not always guaranteed, with up to 5% of messages getting lost or delivered after long period. These problems appear because of the lower priority of SMS than voice communications.

Other texting problems appear when the services provided to the sender are not the same as those provided to the receiver. As an example of these problems, some wireless providers use 7-bit alphabet for texting messages, while others use 8-bit messaging system. Receiving an 8-bit message by a 7bit provider is likely to result in a garbled string of nonsense text for the end user.

3.2.2.1.1.3 Procedure

Once the message is sent from the user's mobile phone, it will be received to the SMSC (Short Message Service Center) which sends a SMS request to the HLR (Home Location Register) to find the roaming customer in order to deliver the message to it. The HLR then will respond to the SMSC with the subscriber status (active or inactive) and where the subscriber is roaming.

If the subscriber is inactive, the SMSC will store the message for a period of time until the subscriber become active .Then, the HLR will send a SMS notification to the SMSC which will deliver the message to the intended receiver. Therefore the SMS is a store and forward service.

3.2.2.2 GPRS Modem

GPRS modem is a GSM modem that additionally supports GPRS technology. GPRS modem supports a set of specified AT commands that enable it to send and receive SMS messages and enable the computer connected to it to communicate over GSM and GPRS networks. Fig (3.3) shows a GSM/GPRS module which consists of a GSM/GPRS modem assembled together with power supply circuit and communication interfaces (like RS-232, USB, etc) for computer.

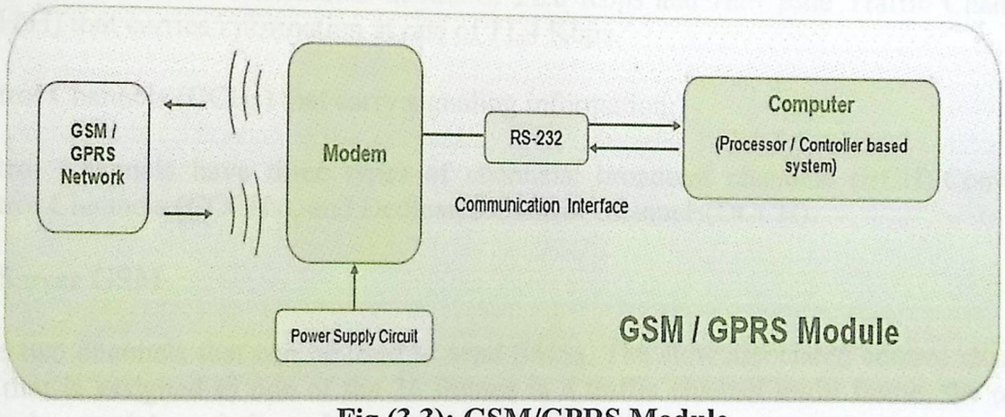


Fig (3.3): GSM/GPRS Module

3.2.2.2.1 GSM Channels

The GSM radio link shares the bandwidth among the users. So it uses both frequency-division multiple access (FDMA) and time-division multiple access (TDMA) for this purpose.

3.2.2.2.1.1 Physical Channels

FDMA divides the frequency bands of a link into a number of separate frequency channels. The GSM-900 system has two frequency bands available:

890 - 915 MHz for the uplink (direction MS to BS) and 935 - 960 MHz for the downlink (direction BS to MS). These frequency bands are divided into 124 pairs of channels with FDMA.TDMA further divides each carrier frequencies into 8 time slots such that each carrier frequency is shared by 8 users.

The 8 time slots constitutes a TDMA frame of length 4.615 ms. Thus each time slot has the length of 0.577 ms. The repetition of one Particular time slot makes up a physical channel for

the user to transmit data (voice or signaling). Hence a physical channel is defined by both the frequency and the TDMA frame time slot number. Each physical channel transmits in a series of short bursts; hence a GSM terminal using a single time slot is only transmitting 1/8 of the time.[27]

3.2.2.2.1.2 Logical Channels

Logical Channels are determined by the information carried within the physical channel. Consequently two kinds of logical channels are defined:

1. Traffic Channels (TCHs) that carry user traffic (voice or data):

   In GSM system two types of traffic channels are used: Full Rate Traffic Channels (TCH\F) that carries information at rate of 22.8 Kbps and Half Rate Traffic Channels (TCH\H) that carries information at rate of 11.4 Kbps.

2. Control Channels (CCHs) that carry signaling information:

   Control Channels have three types of channels: broadcast channels (BCH),Common Control Channels (CCCHs), and Dedicated Control Channels(DCCH).

3.2.2.2.2 SMS over GSM

There are two channels that can be used to send SMSs, The slow associated control channel SACCH that is assigned to one of the 26 frames in a traffic channel multi frame, the other channel is the standalone dedicated control channel (SDCCH).

During a phone call SMSs are sent through SACCH, and otherwise, when the MS is idle, messages are sent through SDCCH. SDCCH is mainly used for signaling during the call set-up phase.

3.2.2.3 Server

Server is a computer which is used to control the GPRS modem using AT command. It is also responsible for getting the data, analyzing the data, converting the message into email form and sending it to the mail server which forwards it to the specific email address. All of these operations will be performed using the appropriate software.

3.2.3 Message Format

The message structure is shown in fig (3.4)

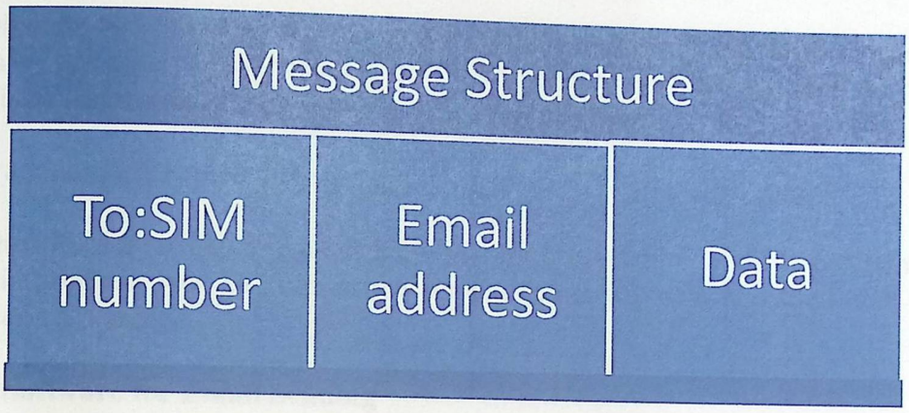


Fig (3.4): Message Structure

The message structure consists of the following:

1. The number of the SIM card that is inserted to the GSM modem.
2. The email address of the recipient followed by space.
3. Data is the message of the user.

The message will be as shown in fig (3.5)

Fig (3.5): Message Format

3.2.4 Software Components

3.2.4.1 Module for Interfacing GPRS modem to the server

Software that will be used for interfacing the GSM modem to the server by determining which serial port is connected to the GSM modem. Then the port will be opened and the data will be transferred.

3.2.4.2 Software for SMS Processing

When the server gets the message, a java program will be used to perform SMS processing. This program will be responsible for extracting the data and the email address from the received message.

3.2.4.3 Software for Email Sending

A java program is also used for sending emails. This program sends emails from a specific Gmail account to other email accounts such as; Yahoo, Hotmail, and Gmail. The extracted email address will be placed as the email address of the recipient and the extracted data will be placed as the email text.

3.3 File to Email Service

3.3.1 System Block Diagram

The system consists of the following basic components:

1. Mobile Phone.
2. Server.
3. Mail server.

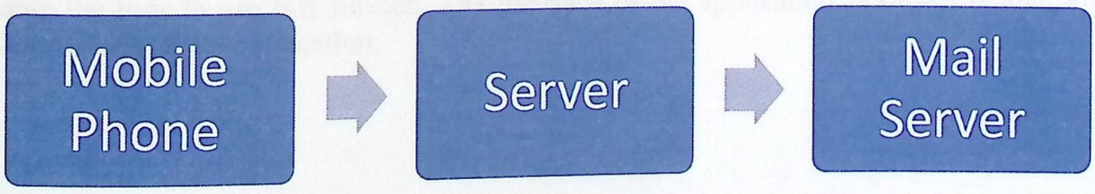


Fig (3.6): Preliminary Design of File to Email Service

A detailed figure illustrating the system entities is shown in fig (3.7)

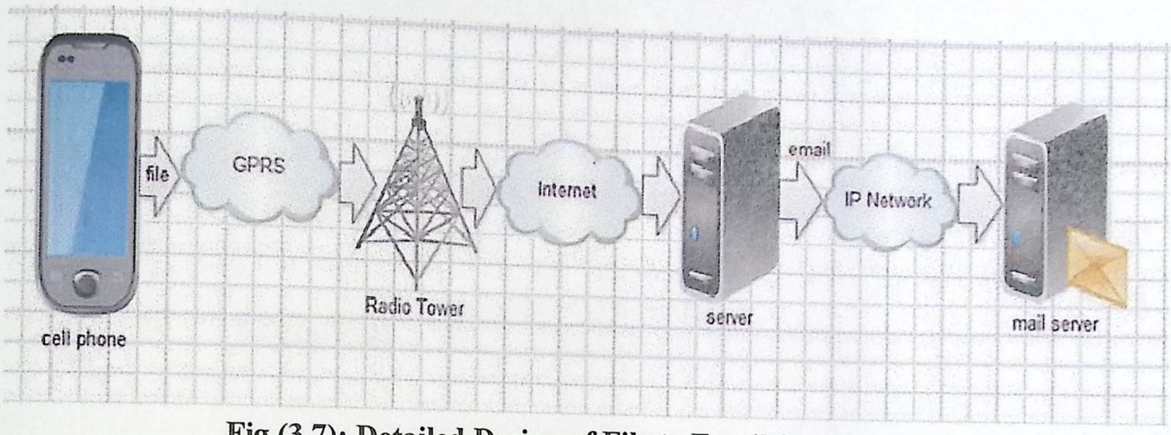


Fig (3.7): Detailed Design of File to Email Service

The user sends the email address of the recipient, file, and text message through the Mobile application from his mobile phone to the radio tower of Jawwal over the GPRS network which sends it to the server over the IP network. The server then will get the data, analyze it, take the email address of the recipient and send it to the mail server which forwards it to the specific Email address.

3.3.2 System Entities

3.3.2.1 Mobile Phone

The mobile phone should support GPRS service. This service enables the users to send files from mobile phone through the GPRS network.

3.3.2.1.1 The mobile application

A mobile application is an android application which can be downloaded on the mobile phone to enable the user to use this service. The interface of the application is shown below. Fig (3.8) shows the mobile application.

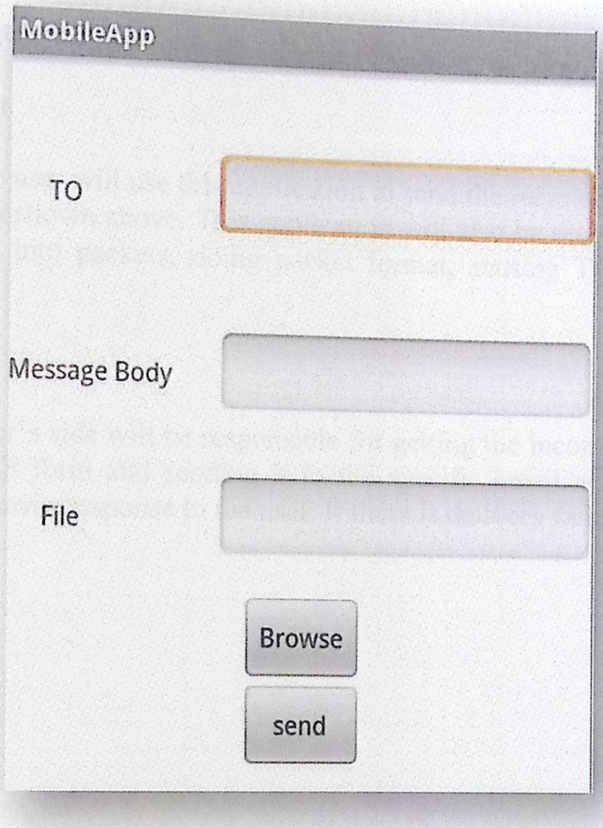


Fig (3.8): The Interface of the Mobile Application.

3.3.2.2 Server

Server is a computer which is responsible for getting the data, analyzing it, converting the data into email form and sending it to the mail server which forwards it to the specific email address. All of these operations will be performed using the appropriate software.

3.3.2.3 Mail Server

A mail server (also known as a mail transfer agent or MTA) is an application that receives incoming e-mail from local users (people within the same domain) and remote senders and forwards outgoing e-mail for delivery. A computer dedicated to running such applications is also called a mail server.

3.3.3 Software Components

3.3.3.1 Mobile Application

On the mobile side, the user will use this application to send the recipients email address , text message and the file as shown above. This application will also be responsible of reading the file, dividing the data into packets, doing packet format, starting TCP/IP connection and sending the data.

3.3.3.2 Server Application

A program on the server's side will be responsible for getting the incoming data, retrieving it, converting it into email form and sending it to the specific email address. It will also be responsible of sending error response to the user, if there is delivery failure.

4.1 Introduction

This chapter presents all the design details. It contains hardware details and software details.

4.2 SMS to Email Service

4.2.1 Hardware Details

Short messages SMS are sent from the mobile phone over GSM network through the SMS application. SMS are sent to the GPRS modem that is connected to the server through the USB port .

4.2.1.1 GSM modem

The GSM modems that we used are 3G HSDPA GSM modem and ADH8066 GSM modem.

4.2.1.1.1 3G HSDPA GSM Modem

3G HSDPA modem base on Siemens HC25 module

Fig (4.1): 3G HSDPA GSM Modem

In 3G HSDPA modem we can use serial port (RS232) which is a port or interface that is used for serial communication, in which only one bit is transmitted at a time, or USB interface.

In this project we used USB interface which is used in a bus for connection, communication and power supply between computers and the modem.

Features of 3G HSDPA modem:

- Supports both Quad-Band GSM/GPRS/EDGE and Tri-Band UMTS/HSDP.

Chapter 4

Detailed Design

4.1 Introduction.
4.2 SMS to Email Service.
4.3 File to Email Service.

4.1 Introduction

This chapter presents all the design details. It contains hardware details and software details.

4.2 SMS to Email Service

4.2.1 Hardware Details

Short messages SMS are sent from the mobile phone over GSM network through the SMS application. SMS are sent to the GPRS modem that is connected to the server through the USB port .

4.2.1.1 GSM modem

The GSM modems that we used are 3G HSDPA GSM modem and ADH8066 GSM modem.

4.2.1.1.1 3G HSDPA GSM Modem

3G HSDPA modem base on Siemens HC25 module

Fig (4.1): 3G HSDPA GSM Modem

In 3G HSDPA modem we can use serial port (RS232) which is a port or interface that is used for serial communication, in which only one bit is transmitted at a time, or USB interface.

In this project we used USB interface which is used in a bus for connection, communication and power supply between computers and the modem.

Features of 3G HSDPA modem:

- Supports both Quad-Band GSM/GPRS/EDGE and Tri-Band UMTS/HSDP.

- A full voice & data support with different voice modes.
- RIL/NDIS/USB driver for devices based on Microsoft Windows Mobile 5.0 and its successor USB 2.0 full-speed interface.
- Fulfill both local and global GSM and UMTS requirements.
- The perfect choice for PDAs, entertainment devices, industrial handhelds and USB modems.

4.2.1.1.2 ADH8066 GSM modem

The ADH8066 is a miniature, quad-band GSM 850/EGSM 900/DCS 1800/PCS 1900 module, which can be integrated into a great number of wireless projects. It can be used to accomplish almost anything a normal cell phone can - SMS text messages, GSM/GPRS, TCP/IP, and more. It also includes a SIM card socket on the back.

Also in this modem we use USB port which is used in a bus for connection, communication and power supply between computers and the modem. Fig(4.2) shows ADH8066 Evaluation Board .Fig (4.3) shows ADH8066 GSM Modem.

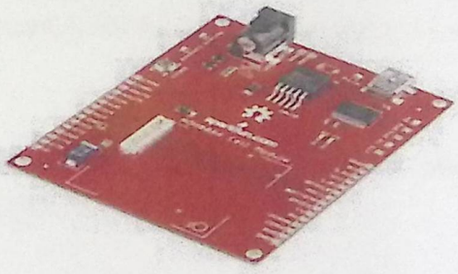


Fig (4.2): ADH8066 Evaluation Board

The evaluation board for the ADH8066 Cell Module breaks out all the pins to the module and provides the necessary support hardware to get up and running. The ADH8066 board attaches to any USB port and appears as a standard com port. Power up the board with 6-12 VDC, turn on the module, and it can start sending and receiving AT commands via HyperTerminal or favorite terminal program.

Fig (4.3): ADH8066 GSM Module

4.2.1.3 USB port:

Universal Serial Bus (USB) is a specification to establish communication between devices and a host controller (usually personal computers). An USB system consists of a host controller and multiple devices connected using special hub devices. Hubs may be cascaded, up to 5 levels. USB can connect computer peripherals such as mice, keyboards, digital cameras, PDA, mobile phones, printers, personal media players, flash drives, GPS, Network Adapters, and external hard drives. For many of those devices, USB has become the standard connection method.

4.2.1.3.1 USB connectors

There are several types of USB connectors. The original USB specification detailed Standard-A and Standard-B plugs and receptacles.

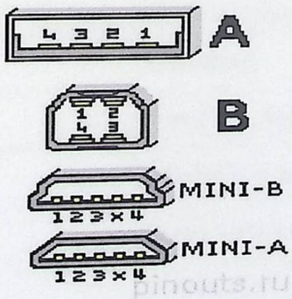


Fig (4.4): 4 pin USB A or USB B Plug Connector

Table (4.1): The USB Pinout

Pin	Name	Cable color	Description
1	VCC	Red	+5 VDC
2	D-	White	Data -
3	D+	Green	Data +
4	GND	Black	Ground

4.2.2 Software Details

4.2.2.1 AT commands

The following AT commands are used for receiving and sending messages through GPRS modem

Table (4.2): AT Commands for Receiving Messages

AT command	Meaning
+CNMI	New message indications
+CMGL	List messages
+CMGR	Read messages
+CNMA	New message acknowledgement

Table (4.3): AT Commands for Sending Messages

AT command	Meaning
+CMGS	Send message
+CMSS	Send message from storage
+CMGW	Write message to memory
+CMGD	Delete message
+CMGC	Send command
+CMMS	More messages to send

4.2.2.2 Main Module

The following flow chart illustrates the main module.

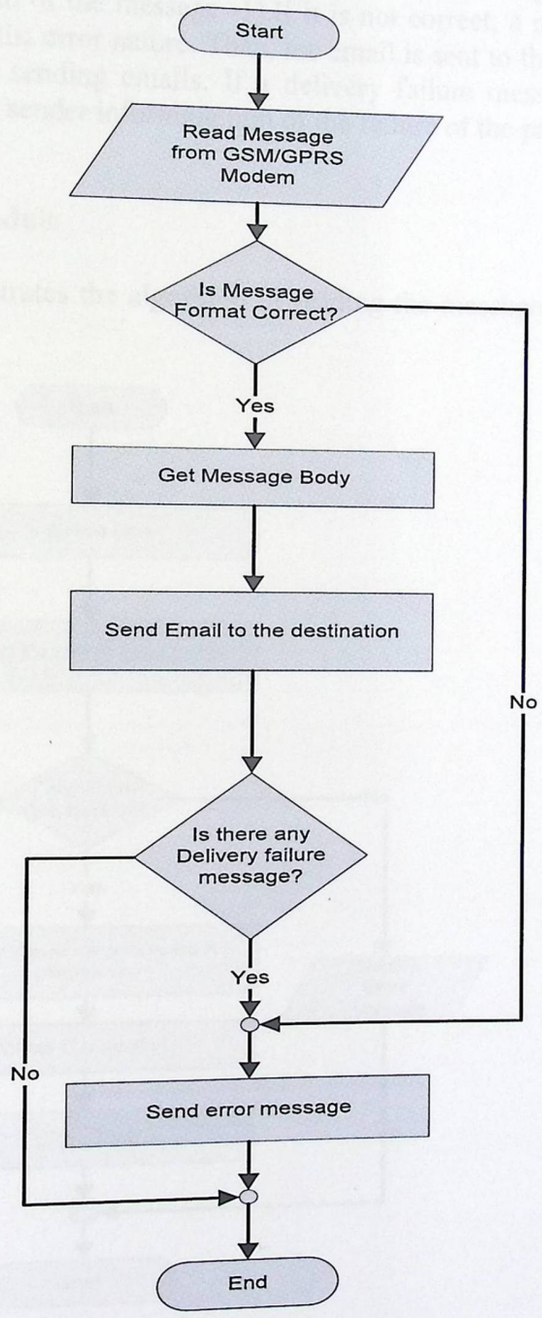


Fig (4.5): Main Module

First the server reads the message from GSM/GPRS modem through AT commands. Second, it checks the format of the message through a special algorithm by taking in consideration the index of the first space, dot, and @ for also having a correct email format .If the message format is correct, we get the message body through defining a substring from (the index of the first space+1) to (the length of the message -1).If it is not correct, a message will be sent to the user informing him of the error nature. Then, the email is sent to the destination through a specific java program for sending emails. If a delivery failure message is arrived, then a message will be sent to the sender informing him of the failure of the process.

4.2.2.3 Reading Message Module

The following flowchart illustrates the algorithm of reading the message from the GSM/GPRS modem.

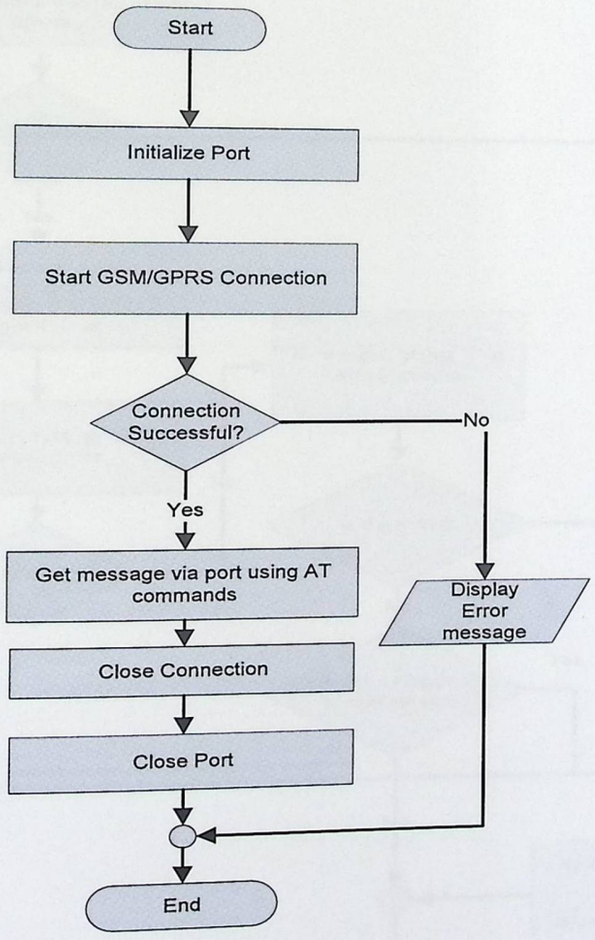


Fig (4.6): Reading Message Module

First the serial port of the computer that is connected to the GSM/ GPRS is initialized. Then, a connection to the GSM/GPRS modem is established to get the data. If there is any problem in the connection, an error message will be displayed. Otherwise the message is obtained through AT commands. After that, the connection is closed and the port is closed.

4.2.2.4 Correct Format Module

The following flowchart illustrates the algorithm of checking the format of the message.

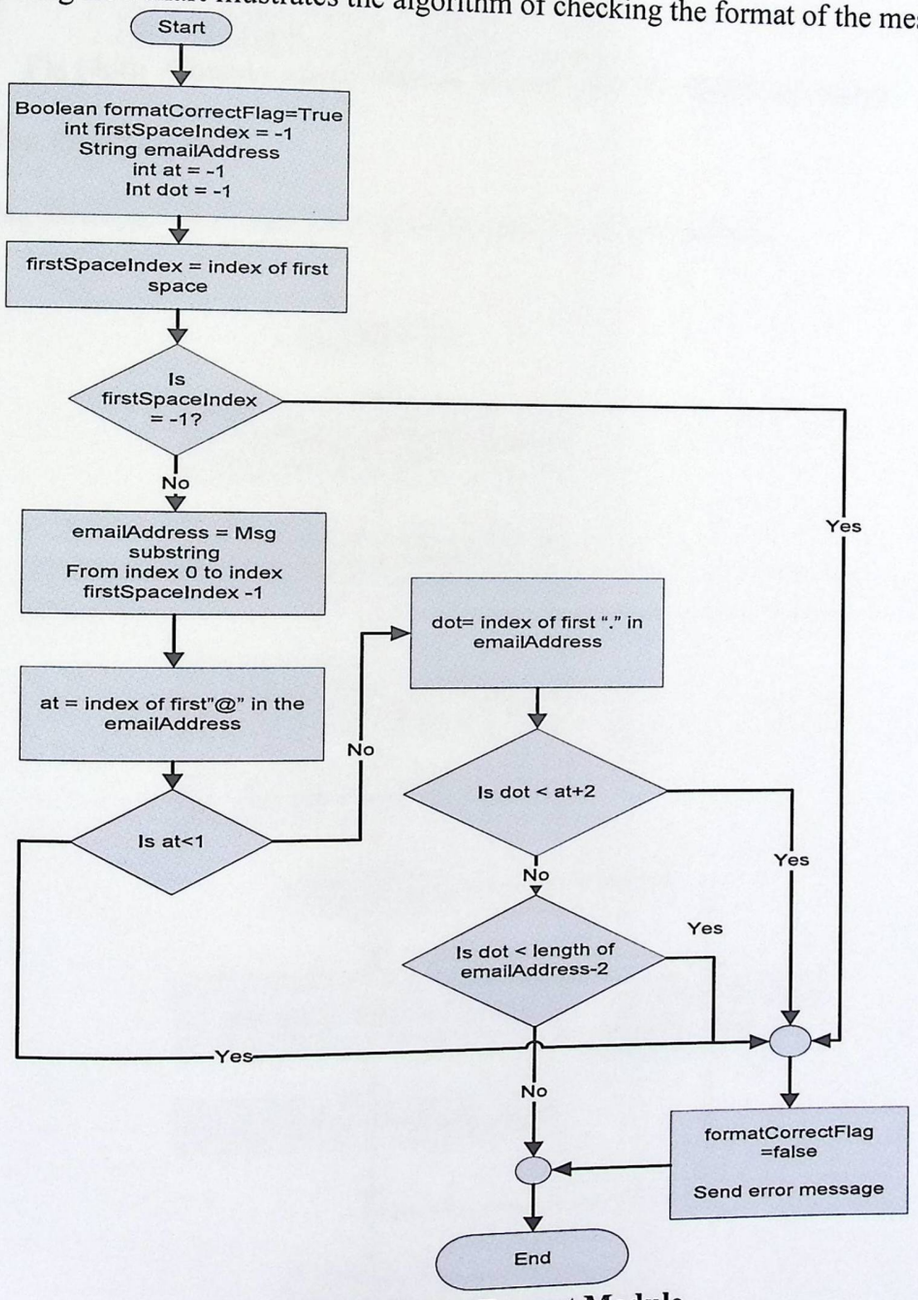


Fig (4.7): Correct Format Module

In this algorithm, the index of the first space is determined. Then, the email address is obtained by defining a substring from (index (0)) to (the index of the first space).The index of @ is determined with taking in consideration that there is at least a character before it. The index of dot"." is determined with also taking in consideration that there are at least 2 characters between @ and dot and 2 characters after dot. If the format is incorrect, an error message will be sent to the sender. Fig (4.8) shows an example of a correct email address format with the minimum length.

A	@	R	J	.	p	S

Fig (4.8): Correct email address format with the minimum length.

4.2.2.5 Sending Email Module

The following flowchart illustrates the algorithm used for sending emails.

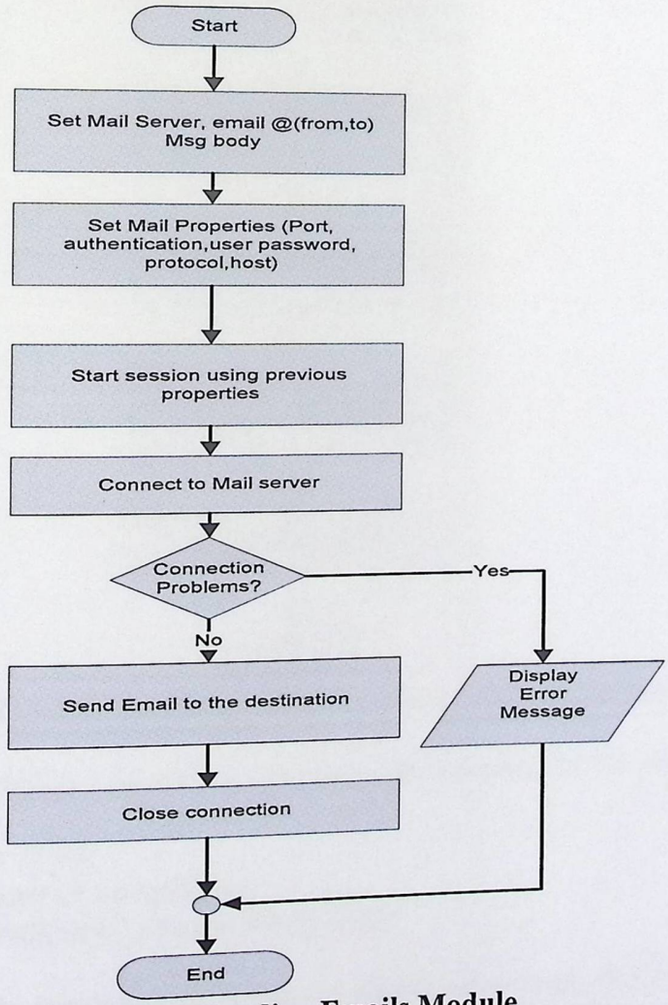


Fig (4.9): Sending Emails Module

For sending emails using this module, first, set the email address of the recipient, the email address of the sender, and the message body. The email address of the sender is a specific Gmail account which will be used for sending emails to any other account. The email address of the recipient is the email address that was extracted from the original message and the message body is the message that was extracted from the original message. Then, we need to set the mail properties such as; port, user, password, authentication, protocol, host. These properties are needed for establishing a session in order to connect to the mail server. If there is any problem in the connection an error message will be displayed. Otherwise, the email will be sent to the intended recipient and the connection will be closed.

4.2.2.6 Reading Delivery Failure Message Module

The following flowchart illustrates the algorithm needed for reading delivery failure message.

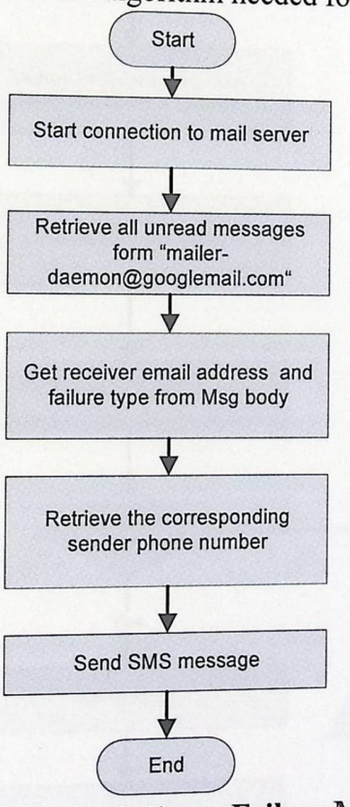


Fig (4.10): Reading Delivery Failure Message Module

Sometimes the email failed to be delivered to the intended recipient for one of the following reasons:

1. The mail server is down.
2. Invalid email address of the recipient.
3. The inbox of the recipient's email account is full.

Then a delivery failure message will be sent to the Gmail account that is used for sending emails. To Know if there is a delivery failure message we need first to connect to the mail

server in order to retrieve all unread messages that is sent from" mailer-daemon@googlemail.com". Then, we get the recipient email address and the failure type from the message body .In order to send a message to the sender informing him of the error nature , the corresponding phone number of sender need to be retrieved from the information of the sent message that is read through GPRS modem. finally the message is sent to the user.

4.2.2.7 Sending Error Message Module

The following flowchart illustrates the algorithm needed for sending error message to the user.

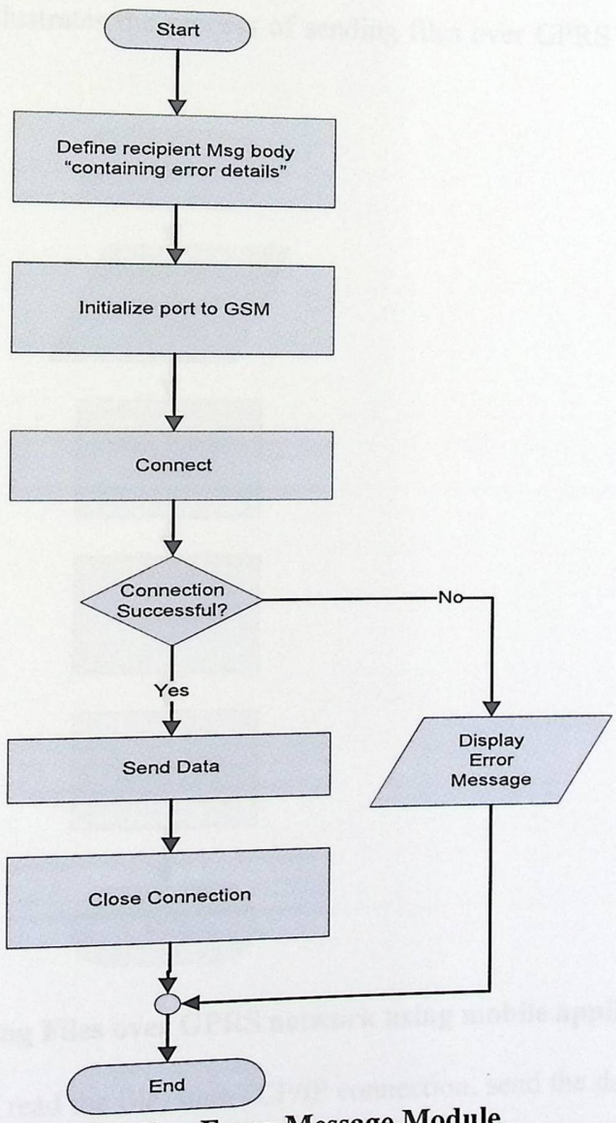


Fig (4.11): Sending Error Message Module

For sending an error message, define the message body with the error type. Second, initialize the serial port that is connected to the GSM/GPRS modem and connect to it. If there is any problem in the connection, an error message will be displayed. Otherwise, the message will be sent and the connection will be closed.

4.3 File to Email Service

4.3.1 Software Details

4.3.1.1 Mobile Application

The following flowchart illustrates the process of sending files over GPRS network through mobile application.

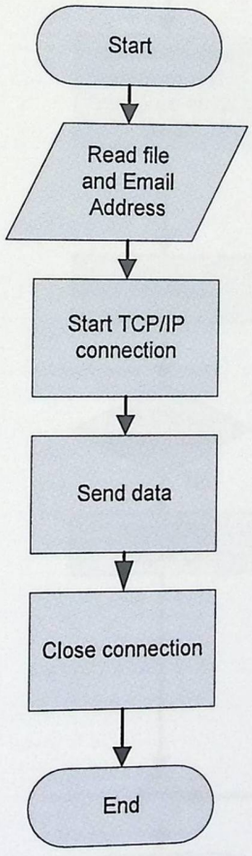


Fig (4.12): Sending Files over GPRS network using mobile application

The mobile application will read the file, start TCP/IP connection, send the data and close the connection.

4.3.1.2 Server Application

The following flowchart illustrates the server application

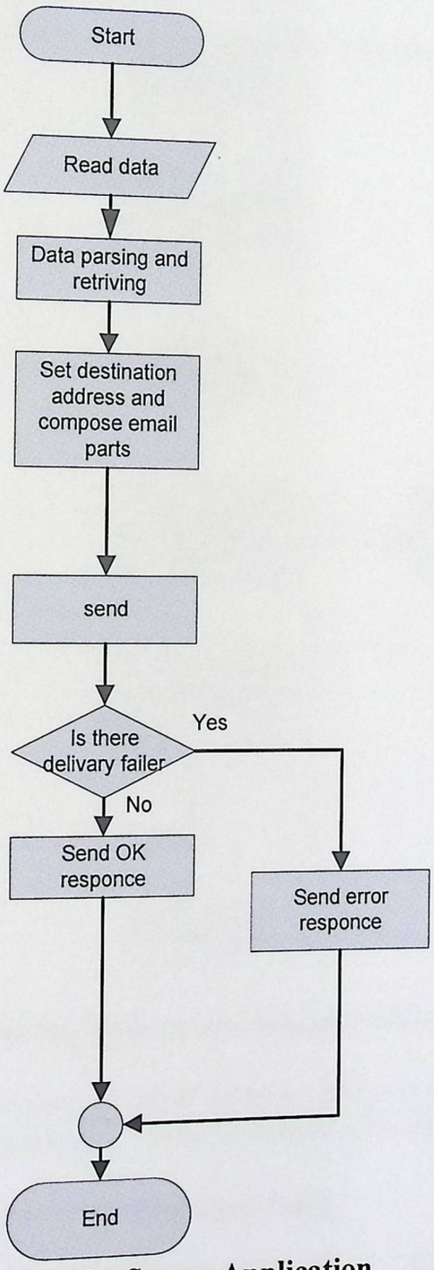


Fig (4.13): The Server Application

The server will read the data, do data parsing and retrieving, set the destination address and compose the email parts, then send the email to the mail server, send OK response. If there is delivery failure, the program sends error response.

4.3.1.3 Storing Data in the Database Module

The following flowchart illustrates the process of storing the information in the database.

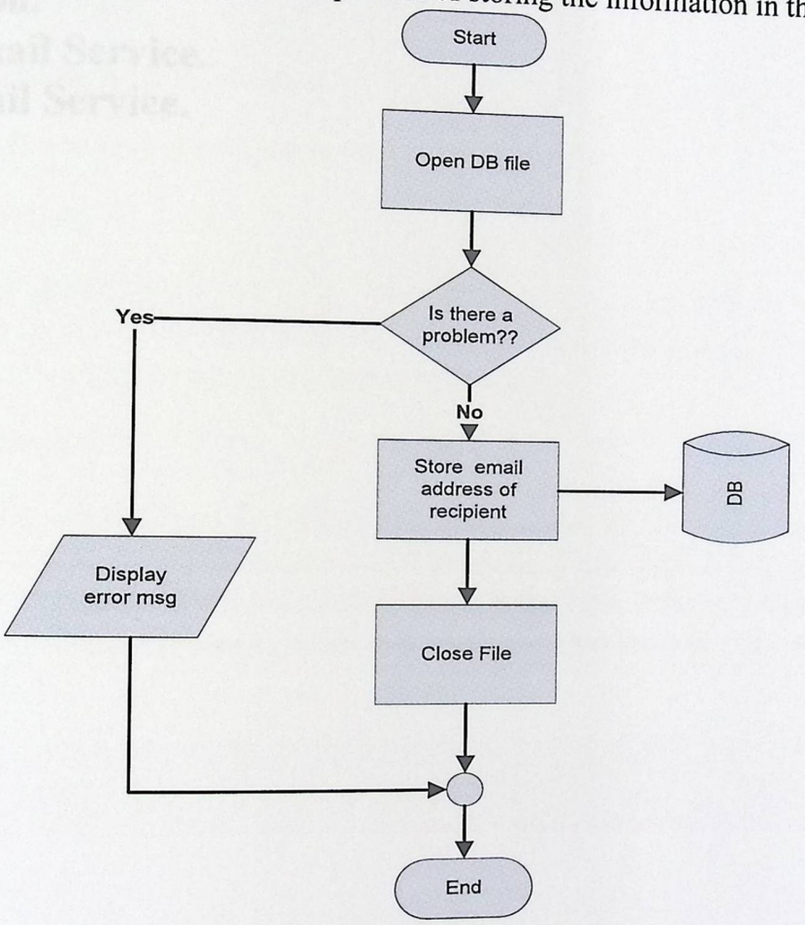


Fig (4.14): Storing Data in the Database Module

The database file contains a table in which the email address of the recipient ,the serial number and phone number of the sender are stored in it. Table (4.4) shows the database table.

Table (4.4): Database Table

SN	PhoneNo	Destination address

Chapter 5

Implementation

5.1 Introduction.
5.2 SMS to Email Service.
5.3 File to Email Service.

5.1 Introduction

This chapter presents all the details about the implementation phase which includes hardware configuration and codes description.

5.2 SMS to Email Service

5.2.1 Implantation Phase

5.2.1.1 Configuration of GPRS modem.

The GPRS modem was configured trough the following steps:

1) Installing the driver.

By installing the appropriate driver, the operating system will be able to recognize to the connected device. A driver typically communicates with the device through the computer bus or communications subsystem to which the hardware connects.

2) Determining the port.

The GPRS modem was configured as a modem that is connected through a specific COM port .This COM port can be known through Device Manager. From Device Manager, we selected Modems tab and then the name of the modem that is connected to the server. The name of the GPRS modem that we used is Sim Tech HS-USB Modem 9000#2. Fig (5.1) shows the Device Manager window.

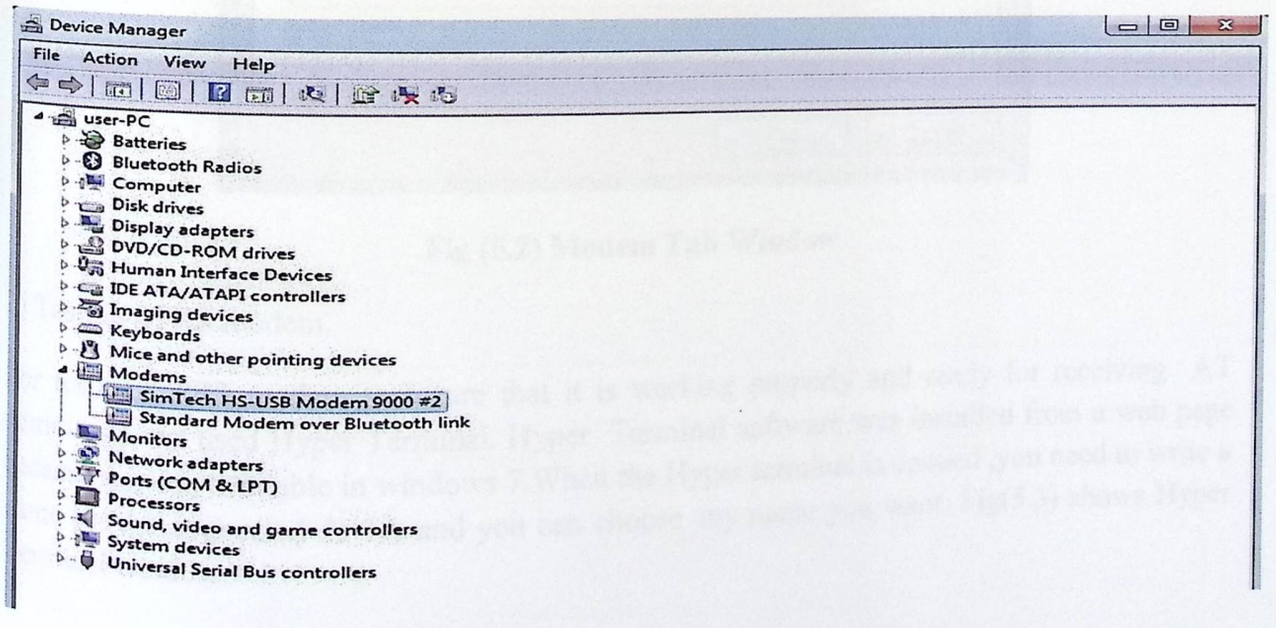


Fig (5.1): Device Manager Window

5.1 Introduction

This chapter presents all the details about the implementation phase which includes hardware configuration and codes description.

5.2 SMS to Email Service

5.2.1 Implantation Phase

5.2.1.1 Configuration of GPRS modem.

The GPRS modem was configured trough the following steps:

1) Installing the driver.

By installing the appropriate driver, the operating system will be able to recognize to the connected device. A driver typically communicates with the device through the computer bus or communications subsystem to which the hardware connects.

2) Determining the port.

The GPRS modem was configured as a modem that is connected through a specific COM port .This COM port can be known through Device Manager. From Device Manager, we selected Modems tab and then the name of the modem that is connected to the server. The name of the GPRS modem that we used is Sim Tech HS-USB Modem 9000#2. Fig (5.1) shows the Device Manager window.

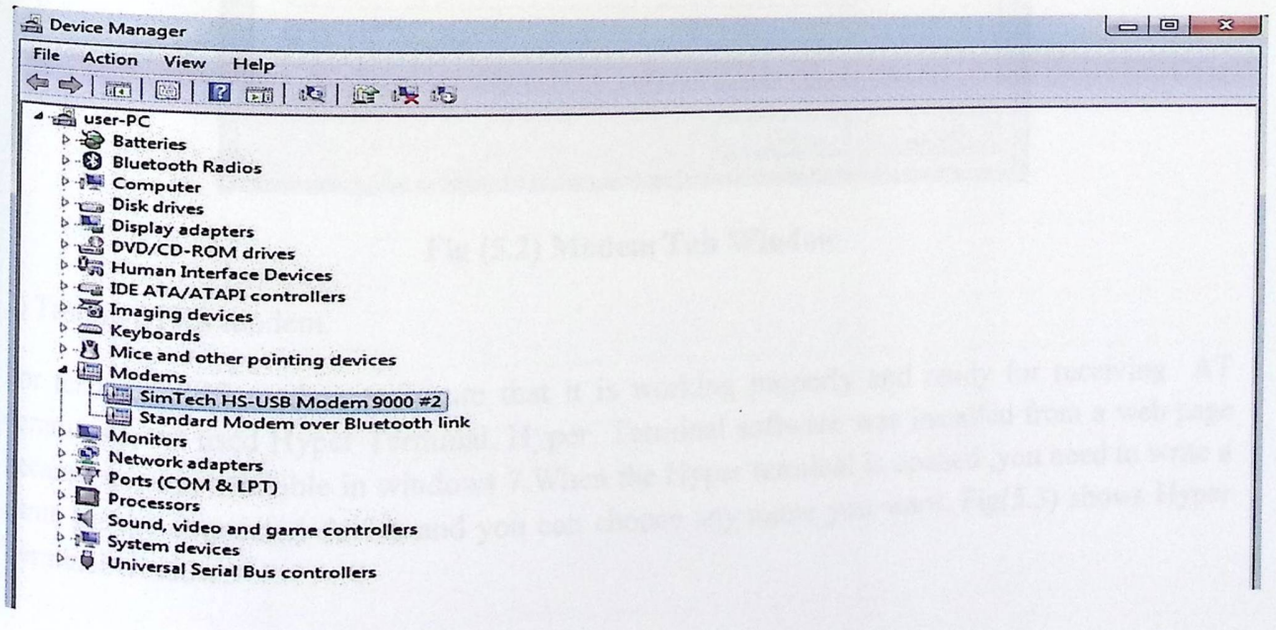


Fig (5.1): Device Manager Window

After selecting the modem, the modem properties window will show up to illustrates all the properties of the connected modem. From modem tab, you can know the specified COM port and the maximum port speed.fig (5.2) shows Modem tab window.

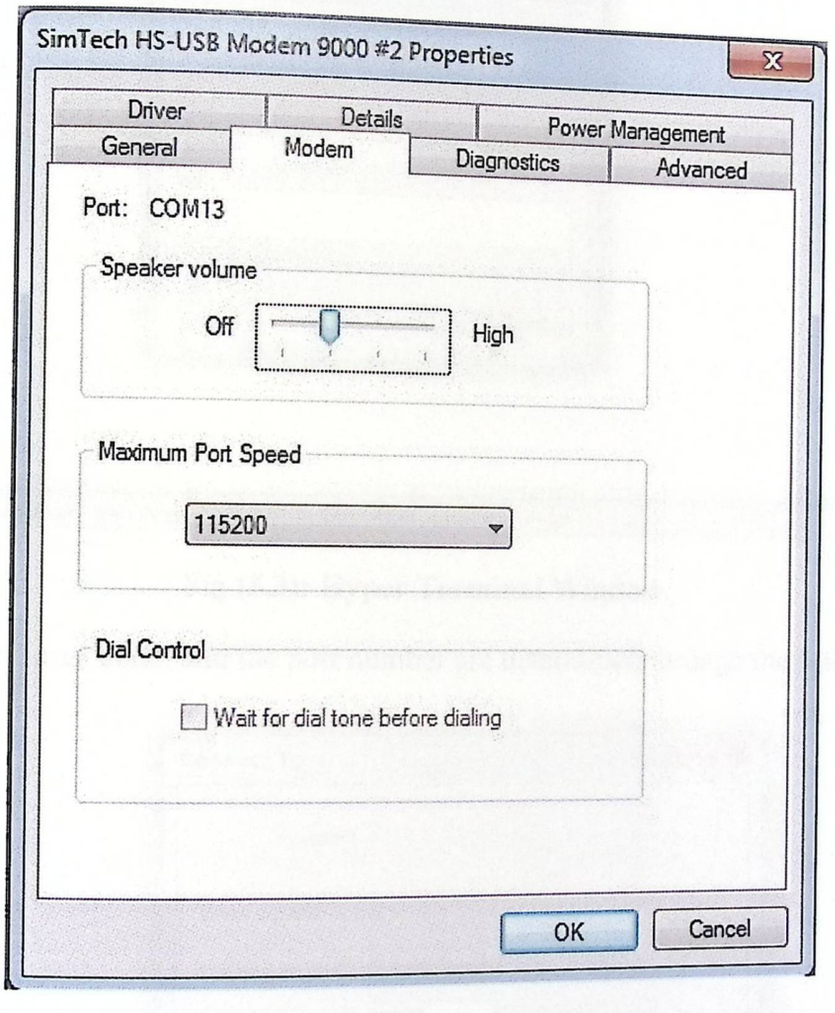


Fig (5.2) Modem Tab Window

3) Testing GPRS modem.

For testing GPRS modem to insure that it is working properly and ready for receiving AT commands, we used Hyper Terminal. Hyper Terminal software was installed from a web page because it is not available in windows 7.When the Hyper terminal is opened ,you need to write a name for the connected device and you can choose any name you want. Fig(5.3) shows Hyper Terminal window.

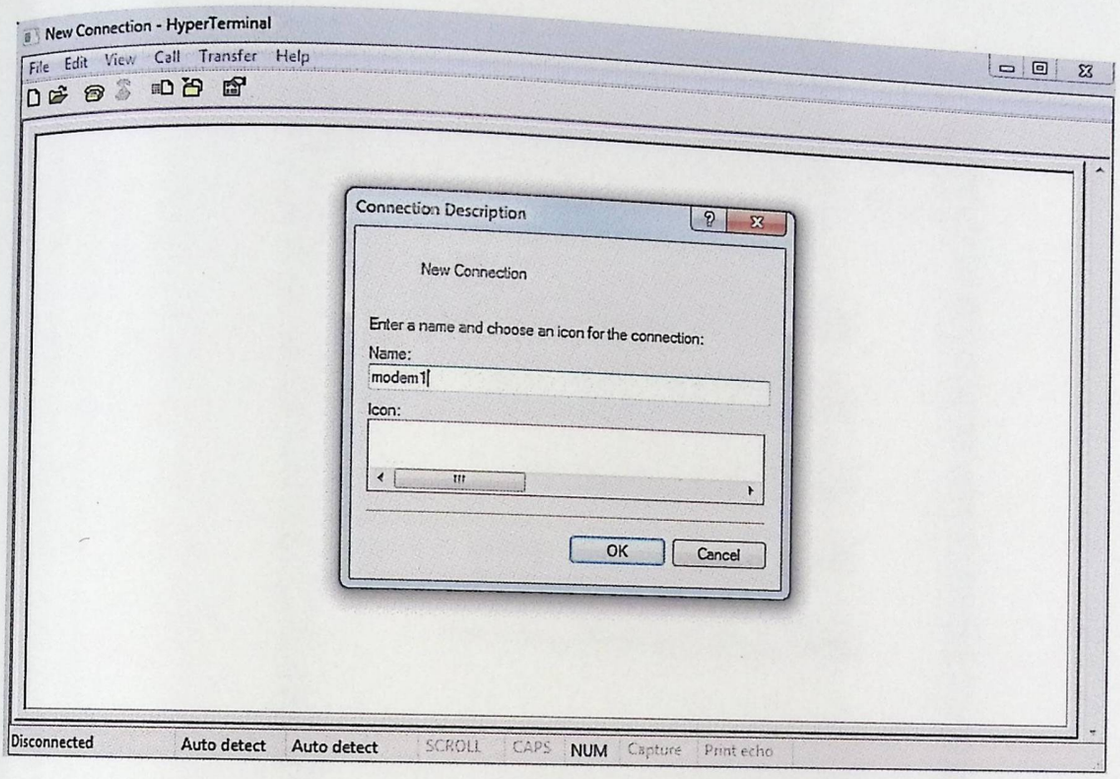


Fig (5.3): Hyper Terminal Window.

Then, the country, area code, and the port number are determined through the following window.

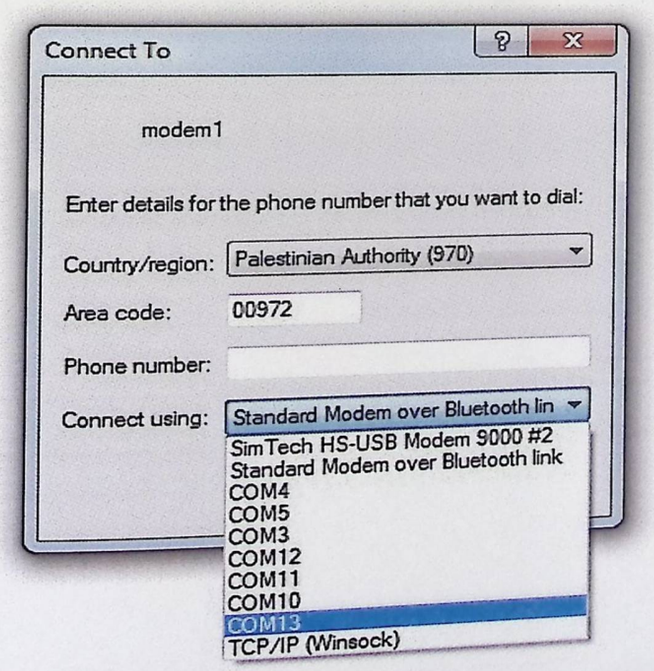


Fig (5.4): Modem1 Window.

Then , COM port properties must be determined through the following window.

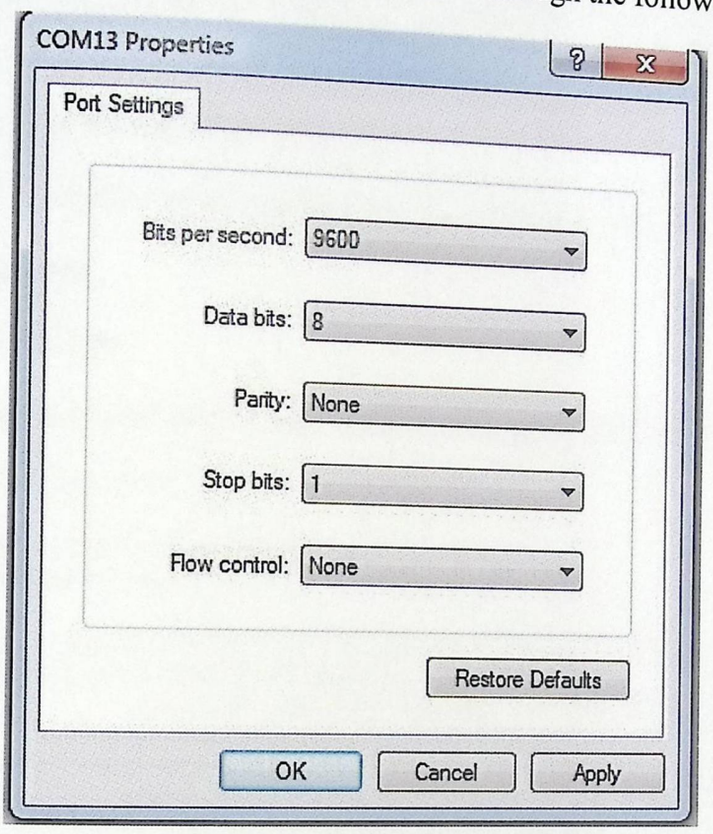


Fig (5.5):COM13 Properties window.

Now, We can write AT commands.

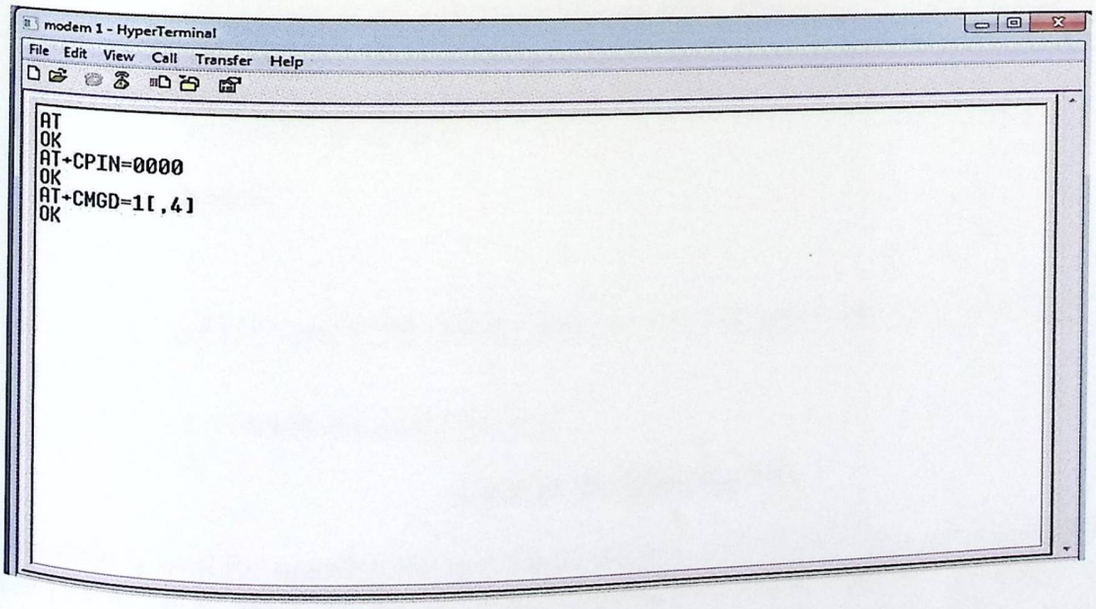


Fig (5.6): AT Commands

"AT "is used to test if the modem is connected successfully or not. If "OK" returns, then the modem is connected successfully .If "ERROR" returns, then the modem is not connected successfully.

"AT+CPIN=0000" is used for entering PIN code.

"AT+CMGD=1[,4] " is used for deleting all the messages that are stored in the modem.

5.2.1.2 Codes Description:

5.2.1.2.1 Opening Port Code

The process of opening the serial port is implemented through the following code:

```
public  static void openPort(){
        try{
           boolean boolPortOk=false;
           strPortName="com13";

            portList=CommPortIdentifier.getPortIdentifiers();
            while(portList.hasMoreElements()){
                        portId=(CommPortIdentifier)portList.nextElement();

            if(portId.getPortType()==CommPortIdentifier.PORT_SERIAL){
                        if(portId.getName().equalsIgnoreCase(strPortName)){

            Main.serialPort=(SerialPort)portId.open("SimpleWriteApp", 2000);

            outputStream=serialPort.getOutputStream();
            inputStream=serialPort.getInputStream();
            serialPort.notifyOnDataAvailable(true);
            serialPort.setSerialPortParams(9600,
            SerialPort.DATABITS_8, SerialPort.STOPBITS_1,
            SerialPort.PARITY_NONE);
            boolPortOk=true;

            break;
            }
            }
            }
            if(!boolPortOk)throw new PortNotReadyException _(strPortName);


            }  catch(Exception e){}
            }
```

Code (5.1): Opening Port

This code is used for opening the specified com port and thus reading and writing the data from and into this port. First, we specified the connected port to be "com13" as shown in the above code.(*CommPortIdentifier*) returns an Enumeration of all the com ports that are available in the

computer .(*getPortType()*) is a method that determines whether the port is serial port or not.Com13 will be selected from the port list which contains all the com ports. After that, the port will be opened so we can get the incoming data through *getInputStram()* method and output data through *getOutputStream()* method.

(*notifyOnDataAvailable(true)*) receives a notification when data is available .(*setSerialPortParameter*) is used to set the port parameters and it takes four parameters:

1. baudrate :9600

2. DataBits: 8.

3. StopBits: 1.

4. Parity: none.

9600 baud rate , 8 data bits, 1 stop bits, and no parity is the default parameters for the serial port.

5.2.1.2.2 Reading SMS code

The process of reading SMS is implemented through the following code:

```
public static  String readSMSByIndex(int index){
      try{
            final String INIT_ME="AT+CPMS=\"ME\"\r";
            final String TXT_MODE="AT+CMGF=1\r";
            final String AT_List="AT+CMGR="+index+"\r";
            StringBuffer sb=new StringBuffer();
            writeATCmd(INIT_ME);
            Thread.sleep(500);
            writeATCmd(TXT_MODE);
            Thread.sleep(500);
            sb.append(writeATCmd(AT_List));
            Thread.sleep(500);
            return sb.toString();
         }
      catch(Exception ex)
        {
              System.out.print(ex.getMessage());
              return "err";
         }

       }
       }
```

Code (5.2): Reading SMS

The shown code is used for reading SMS by specifying the index of it. AT commands were sent to the modem through (*writeATCmd)* method. Here are the AT commands:

1. *"AT+CPMS=\"ME\"\r"* is used for setting the message storage at ME (mobile Equipment) .

2. *"AT+CMGF=1\r"* is used for setting message format to the text mode .if we replace "1" by "0" then the PDU mode will be implemented.

3. *"AT+CMGR=""+index+"\r"* is used for reading message with the specific index from the message storage.

For reading all the messages we used a counter (index++) which starts from 0 and a timer which checks if there is a new message every 7 seconds.

5.2.1.2.3 Checking Message Format Code

The process of checking the message format is implemented through the following code:

```
public static  void formatCheck(String str, String mobileNumber){
   boolean invalidFormatFlag= false;
   String mailAddress="", msg="";
   try{
      int firstSpacePosition;
      firstSpacePosition = str.indexOf(" ");
      if (firstSpacePosition<0){
          invalidFormatFlag = true;
          System.out.println("Invalid Format space");
        }else{
          mailAddress  = str.substring(0, firstSpacePosition);
          if(mailAddress.indexOf('@')<1){
             invalidFormatFlag = true;
             System.out.println("Invalid Format @");
            }else{
                String
            s=mailAddress.substring((mailAddress.indexOf('@')),
            mailAddress.length());
            if(s.indexOf('.')<(s.indexOf('@')+3) ||
            mailAddress.indexOf('.')>= mailAddress.length()-2){
                     invalidFormatFlag = true;
                     System.out.println("Invalid Format dot");
                   }
                   else{
                     msg = str.substring(firstSpacePosition+1);
                     System.out.println("Address: " + mailAddress);
                     System.out.println("Message: " + msg);
               }
               }}
```

Code(5.3): Checking Message Format

The shown code is used for checking the format of the message if it is a valid format or not. Valid format of the message as we determined in chapter 3 begins with the email address of the recipient and then a space followed by the message that the user wants to send as email.

First, the position of the first space is determined .If the space doesn't exist then the message format is invalid. Otherwise, the email address will be extracted from the original message. Email address will be defined as a substring that begins from index (0) to index of the first space. After that, we determined the index of (@) .if it is <1 which means that @ either doesn't exist or comes at the beginning of the message, then the message format is invalid. Following that, the index of (.) will be determined. If index of (.) is less than (index of (@)+3) or greater than or equals(email address length -2) then the message format is invalid. That means that there must be at least 2 characters between (@) and (.) and two after (.).

When the message format is valid, the message will be extracted from the original message through defining a substring from index of (first space+1) to the end of the message.

If the message format is invalid, a message will be sent to the user to inform him of that.

After extracting the email address and the message body, the email will be sent through a code for sending emails.

5.2.1.2.4 Sending Notification of the Delivery Failure Code

The process of sending notification of the delivery failure is implemented through the following steps:

1. Opening the Gmail account that is used for sending emails

2. Reading unread emails that are sent from mailer-daemon@googlemail.com. These emails are delivery failure emails. The Reading process is implemented through a code for reading the unread messages from the Gmail account.

3. Extracting the email address of the intended recipient from the content of the message. The phone number is obtained from the read message that was sent to the modem from the user.

4.Deleting the delivery failure emails from the inbox.

5. Sending SMS to the user to inform him of the delivery failure. The sent message is as shown:

Delivery to the following recipient failed permanently:aaaaa@yahoo.com.

Code of reading delivery failure emails is shown below

```
Properties props=System.getProperties();
props.setProperty("mail.store.protocol", "imaps");

try{
    Session session=Session.getDefaultInstance(props,null);
    Store store=session.getStore("imaps");

   store.connect("imap.gmail.com",sendmail.getUserID(),sendmail.getPassword()
   );
    System.out.println(store);
    Folder inbox=store.getFolder("Inbox");
    inbox.open(Folder.READ_WRITE);
    FlagTerm ft=new FlagTerm(new Flags(Flags.Flag.SEEN),false);
    Message messages[]=inbox.search(ft);
    for(Message m:messages){
            if(m.getFrom()[0].toString().equals("Mail Delivery
            Subsystem<mailer-daemon@googlemail.com>")){
             String content = m.getContent().toString();
             System.out.println("Content >> " + content);
             to2=content.substring(64, content.indexOf(".")+4);
             System.out.println(to2);
             msg2="Delivery to the following recipient failed
            permanently:"+to2;
             sendSMS(moNumb,msg2);
             m.setFlag(Flags.Flag.DELETED, true);

           }
           }
             inbox.close(true);
             store.close();

                    }
catch(MessagingException e){
      e.printStackTrace();
      System.out.println("abc");
      System.exit(1);

       }
catch(Exception e){
      e.printStackTrace();
      System.exit(2);
         }

          }
catch(Exception e){
   }

    }
}
```

Code (5.4): Reading Delivery Failure Emails

5.3 File to Email Service

5.3.1 Implantation Phase

5.3.1.1Application Interface

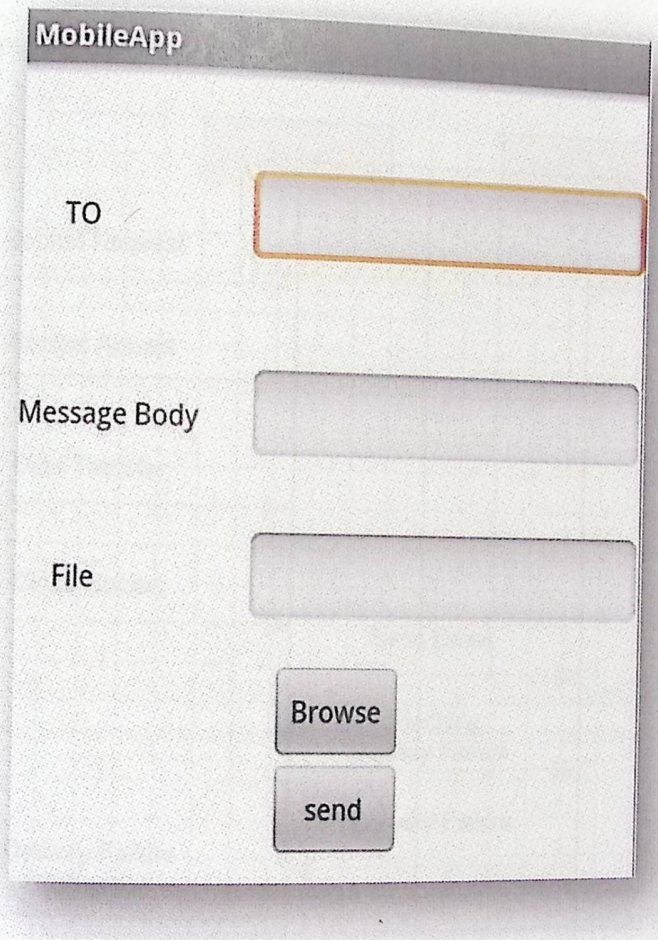


Fig (5.7): Application Interface

As it is shown in the application interface, there are three fields and two buttons:

1."To" field: is used for inserting the recipient email address.

2."Message Body" field: is used for inserting the message text.

3."File" field: is used for showing the path, name, and extension of the file.

4."Browse" button: is used for viewing all the files in the mobile phone and enabling the user to select one of these files for sending it.

5."send" button: is used for sending the file byte by byte through GPRS network to the server which sends it to the mail server which forwards it to the intended email address.

5.3.1.2 Sequence Diagram:

Fig (5.8) shows the sequence diagram of file to email service.

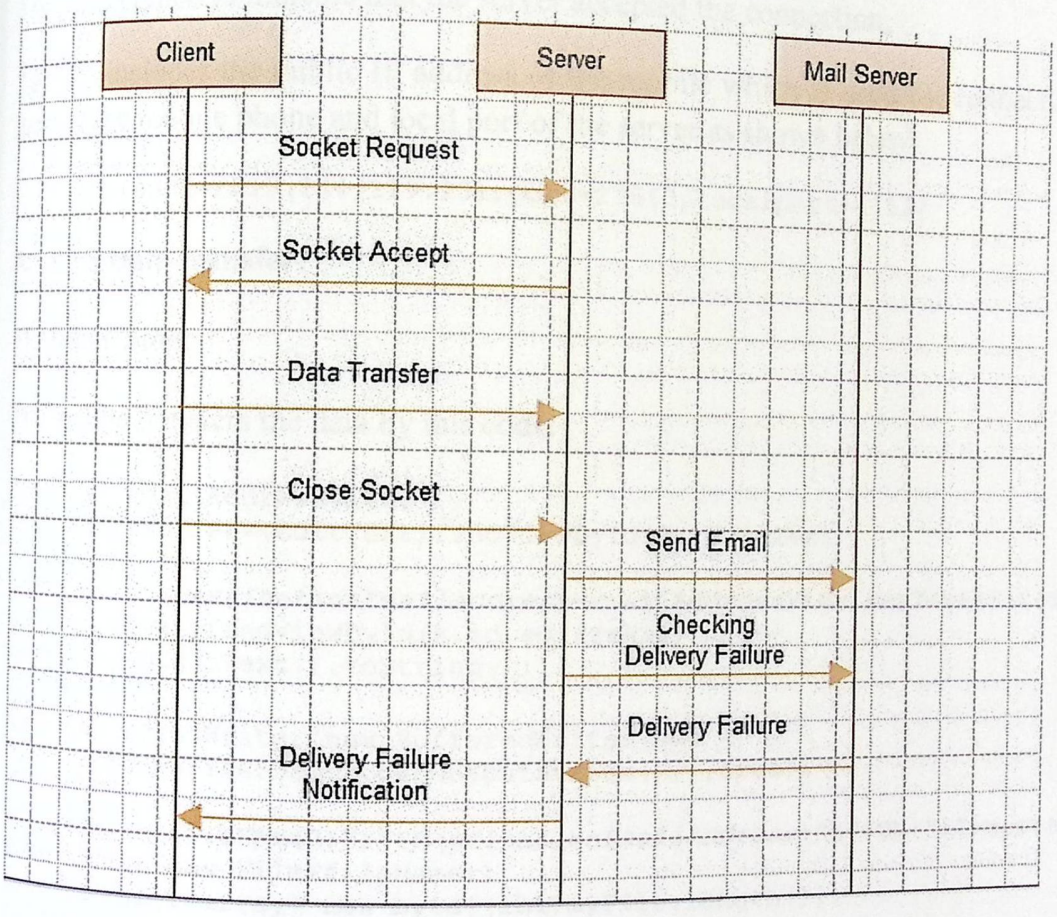


Fig (5.8): Sequence Diagram

5.3.1.2.1 Socket Request:

The mobile instantiates a socket by defining:

1) The public IP of the server.
2) The port number 1112 that is used to enable the client to connect to the server.

```
String IP="195.3.190.16";
sock=new Socket(IP,1112);
```

5.3.1.2.2 Socket Accept:

When any mobile tries to connect to the server through the specified port, the server accepts the connection and opens the socket through this command:

```
sock = servsock.accept();
```

The server receives the data through the input stream. The sentence `("Accepted connection:" +sock)` is typed to indicate that the server accepted the connection.

`"sock"` includes the public IP address of the mobile which is used for connecting to the GPRS ,port of the mobile phone and local port of the server as shown below:

```
Socket[addr=/213.244.119.251,port=27643,localport=1112]
```

5.3.1.2.3 Data transfer:

1) Mobile Side:

The mobile transfers the data by this code:

```
EditText emailAddressView;
emailAddressView=(EditText)findViewById(R.id.toText);

          destinationEmailAddress=emailAddressView.getText().toString();
msg=(EditText)findViewById(R.id.editText1);
message=msg.getText().toString();

out= new PrintWriter(new BufferedWriter(new
OutputStreamWriter(sock.getOutputStream())),true);

out.println(fileName2+";"+destinationEmailAddress+"&"+SN+"$"+message);
File myFile=new File(fileName);
byte [] mybytearray= new byte[(int)myFile.length()];
FileInputStream fis = new FileInputStream(myFile);
BufferedInputStream bis = new BufferedInputStream(fis);
bis.read(mybytearray, 0, mybytearray.length);

OutputStream os= sock.getOutputStream();
System.out.println("Sending...");
os.write(mybytearray, 0, mybytearray.length);
os.flush();
```

Code (5.5): Sending Data from Mobile

In the above code we defined "TO" field for the Email address , "Message Body" field for the message text , and "File" field for the file name .After that, we defined PrintWriter (out) to send

the file name, the destination Email address, serial number of the mobile phone and the message body to the server. BufferedInputStream bis was defined so we can read the file from the array and write it in the buffer (bis) through the following command: `bis.read(mybytearray, 0, mybytearray.length);`.
An output stream was defined so we can send the file byte by byte to the server through this command: `os.write(mybytearray, 0, mybytearray.length);`

2) Server Side

Part1:

```
public void run(){
    try{
        ServerSocket servsock= new ServerSocket(port);
    while(true){
        System.out.println("Server is waiting for clients...");
    do{
        sock = servsock.accept();

    }while(sock==null);
          BufferedReader in = new BufferedReader(new
          InputStreamReader(sock.getInputStream()));
          System.out.println("Accepted connection:" +sock);
          String to1=in.readLine();
```

Code (5.6): Receiving the First Part of the Sent Data

This code is used for receiving data in the server side. First, a constructor (ServerClass(int port)) that takes the port number as an argument was defined. When we call the constructor, the port number must be specified. We called the constructor in the Main class and port number was defined to be 1112 as it shown in the following code.

```
int i=1112;
tc=new ServerClass(i);
```

After that, a new socket was initiated which takes the same port number so that the mobile can connect to the server and start sending data. When we run the server, "Server is waiting for clients...." is typed on the console window to show that the server is ready for accepting the connection from the mobile. The server accepted the connection.

BufferedReader (in)was defined to read the data that was sent from the mobile. The data include file name, email address, serial number, and the message body. The data is read through the following command:

```
String to1=in.readLine();
```

Part2:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
String dataSourceName="database";
String dbURL= "Jdbc:odbc:" +dataSourceName;
Connection con = DriverManager.getConnection(dbURL, "", "");
Statement s= (Statement)con.createStatement();
s.execute("select PhoneNo from Table1 where SN='"+sn+"'");
ResultSet b=s.getResultSet();
 if(b.next())
   String phoneNo=b.getString("PhoneNo");
con.close();
```

Code (5.7): Selecting Phone Number from Database

The above code is used to connect the server to the database in order to select the phone number that relates to the corresponding serial number to send it with the Email.

The following table illustrates the database:

Table(5.1): Database of File to Email Service

SN	PhoneNo	Destination
89970281233268208605	00972598803775	

Part3:

```
byte[]mybytearray = new byte[filesize];
InputStream is = sock.getInputStream();
File tempFile = new File("d:\\"+f);
if(tempFile.exists())
      tempFile.delete();
FileOutputStream fos = new FileOutputStream("d:\\"+f);
BufferedOutputStream bos = new BufferedOutputStream(fos);
while ((count = is.read(mybytearray)) > 0)
                  {
                        System.out.println("bba" + count);
                        bos.write(mybytearray, 0, count);

                  }
bos.close();
fos.close();
```

Code (5.8): Receiving the second part of the data which is the sent file.

The above code illustrates the process of receiving the file from the mobile phone. First, we defined an array with size equals to the size of the file. Input stream was defined for reading the file byte by byte. The file will be saved at the server on Local Disk(D:) through the output stream

.If the file is already exists on(D:) , it will be deleted. After finishing the process of reading the bytes, the buffered output stream and the file output stream will be closed.

5.3.1.2.4 Close Socket:

Finaly socket will be closed by mobile and server using this command:

```
sock.close();
```

5.3.1.2.5 Sending Email:

The process of sending emails is implemented through the following code:

```
Properties props= System.getProperties();
props.put("mail.smtp.starttls.enable","true");
props.put("mail.smtp.host", host);
props.setProperty("mail.transport.protocol",  "smtps");
props.put("mail.smtp.user", userid);
props.put("mail.smtp.password", password);
props.put("mail.smtp.port", "465");
props.put("mail.smtps.auth","true");
Session session = Session.getDefaultInstance (props, null);
MimeMessage message= new MimeMessage(session);
InternetAddress fromAddress = null;
InternetAddress toAddress= null;

try{
   fromAddress= new InternetAddress(from);
   toAddress= new InternetAddress(to);
   }
catch(AddressException e)
          {
    e.printStackTrace();
          }
message.setFrom(fromAddress);
message.setRecipient(RecipientType.TO,toAddress);
message.setSubject(subject);
```

Code (5.9): Sending Email

The above code illustrates the process of sending the Email from the server to the mail server. After defining the host, userid and password of the Gmail account that is used for sending emails , the mail properties such as; port, user, password, authentication, protocol, host were defined. These properties are needed for establishing a session in order to connect to the mail server. Then, we defined the email address parameters which are: from,to ,subject,message body with the file as an attachment.Finally, the email is sent and the connection is closed.

.If the file is already exists on(D:) , it will be deleted ...
bytes, the buffered output stream and the file output ...

5.3.1.2.4 Close Socket:

Finaly socket will be closed by mobile and server using this ...

```
sock.close();
```

5.3.1.2.5 Sending Email:

The process of sending emails is implemented through the following code:

```
Properties props= System.getProperties();
props.put("mail.smtp.starttls.enable","true");
props.put("mail.smtp.host", host);
props.setProperty("mail.transport.protocol", "smtps");
props.put(
props.put(
props.put(
props.put(
Session session
MimeMessage message
InternetAddress
InternetAddress

try{
   fromAddress
   toAddress
   }
catch(Address

     e.print
     }

message.set
message.set
message.set
```

... account and read the delivery ... address of the recipient from the ... by setting the email address in the ... phone number is equal to the phone

Toast

... web application is implemented through the

```
...ext(this,"Waiting for response",
```

The above code illust...
After defining the ho...
, the mail propert...
These properties are...
Then, we defined the...
the file as an attach...

```
...MCAT_PORT+"/WebApplication1/NewServlet1?sn="+SN);
...on) url.openConnection();
...ress=readStream(con.getInputStream());
```

Code (5.11): Mobile Connects to a Web Application

5.3.1.2.6 Checking Delivery Failure

5.3.1.2.6.1 Reading Delivery Failure Emails

Code(5.4) that is used in SMS to Email service is the same code that is used for the second service.

5.3.1.2.6.2 Updating the database

The process of updating the database is implemented through the following code:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
String dataSourceName = "database";
String dbURL = "Jdbc:odbc:" + dataSourceName;
Connection con = DriverManager.getConnection(dbURL, "", "");
Statement s = (Statement) con.createStatement();
s.execute("UPDATE Table1 SET Destination = '" + to2
+ "' where  PhoneNo='" + PNo + "'");
con.commit();
con.close();
```

Code (5.10): Updating the Database

When the server connects to the mail server and opens the gmail account and read the delivery failure emails, it extracts the phone number and the email address of the recipient from the content of the email. The database table will be updated by setting the email address in the Destination field in the same row where the extracted phone number is equal to the phone number that is originally stored at the database.

5.3.1.2.7 Delivery failure Notification

5.3.1.2.7.1 Delivery failure Notification by Toast

The process of connecting the mobile to the web application is implemented through the following code:

```
HttpURLConnection con=null;
Toast makeText = Toast.makeText(this,"Waiting for response",
Toast.LENGTH_LONG);
 try{
makeText.show();
URL url=new
URL("http://"+IP+":"+TOMCAT_PORT+"/WebApplication1/NewServlet1?sn="+SN);
con = (HttpURLConnection) url.openConnection();
String destinationAddress=readStream(con.getInputStream());
```

Code (5.11): Mobile Connects to a Web Application

In the above code, the mobile connects to the web application that contains the servlet program through HttpURLConnection . IP,Tomcat port,web application name and servlet name were determined to set the connection. Also the mobile sent its serial number as a parameter. The servlet program is shown in the following code:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
String dataSourceName="database";
String dbURL= "Jdbc:odbc:" +dataSourceName;

Connection con = DriverManager.getConnection(dbURL, "", "");
Statement s= ( Statement)con.createStatement();
sn=""+request.getParameter("sn").toString();
String x="select Destination from Table1 where SN='"+sn+"'";

s.execute(x);
ResultSet b=s.getResultSet();

  while(b.next()){

     response.getWriter().write(b.getString("Destination"));
    }
s.execute("UPDATE Table1 SET Destination = '' where SN='"+sn.trim()+"'");
con.close();
```

Code (5.12): Selecting Destination from the Database

The above code is used for selecting the destination (email address of the recipient) that was extracted from the delivery failure email. This destination is also related to the serial number that was sent from the mobile as a parameter. In other words, the destination that corresponds to the specified serial number (in the same row) will be selected from the database table and will be returned to the mobile. After that, the destination will be deleted from the database. Then, a toast of the delivery failure that contains the destination will appear at the mobile to inform the user of the delivery failure.

A timer was set to repeat the process of the connection to the database to get the failures for every 2 seconds. The timer will be stopped when the destination is returned to the mobile. In other word, when there is a delivery failure. If there is no delivery failure(Destination=null) the timer will be stopped after 20 seconds and a toast will appear at the mobile to inform the user that the email was sent successfully.

5.3.1.2.7.2 Delivery Failure Notification by Sending SMS:

In addition to the toast that appears at the mobile in the case of the delivery failure, an SMS also will be sent to the user to inform him of the delivery failure. Fig (5.9) and fig(5.10) illustrate the whole process.

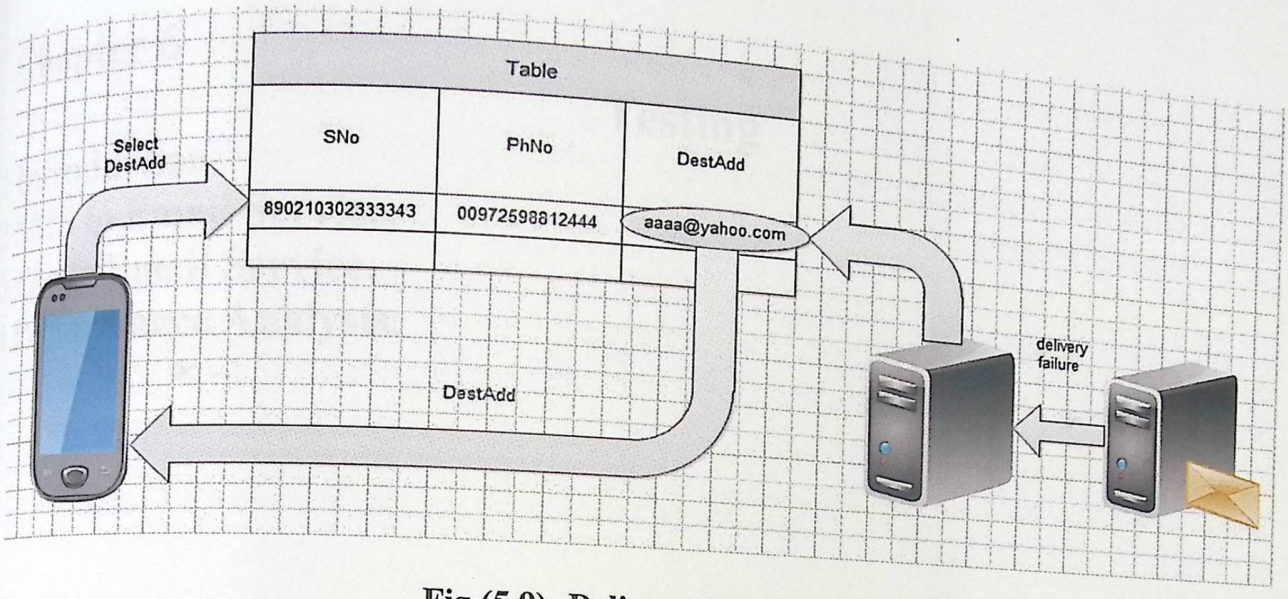


Fig (5.9): Delivery Failure

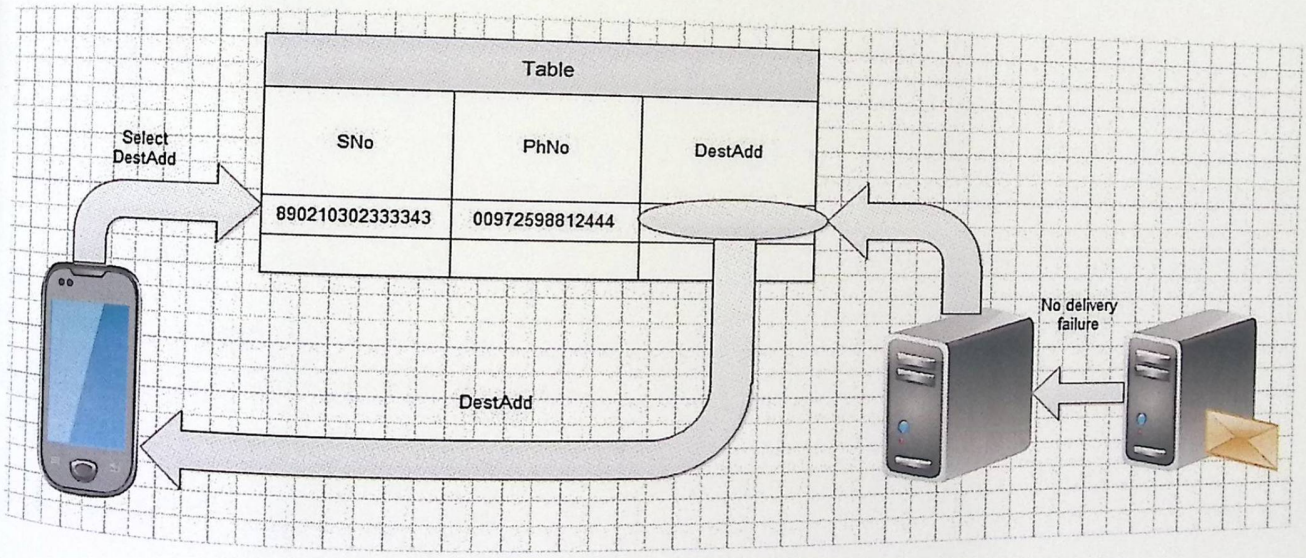


Fig (5.10): No Delivery Failure

First, the server connects to the mail server and read the delivery failure emails. If there is a delivery failure email, it will read the content of the email and extract the email address from it. The extracted email address will be inserted in the destination field at the database. The mobile connects to the database to check if there is a delivery failure. The destination will be returned to the mobile and appears as a toast "Delivery to the following recipient failed permenantly:aaaa@yahoo.com". If there is no delivery failure, a toast "Email was sent successfully" will appear.

Chapter 6

Testing

6.1 Introduction.
6.2 SMS to Email Service.
6.3 File to Email Service.
6.4 Performance Analysis.

6.1 Introduction

This chapter presents the testing phase which includes the results, the success and failure rates

6.2 SMS to Email Service

6.2.1Testing Phase

6.2.1.1 Reading SMS by the server

Fig(6.1) shows the process of reading the received messages at the server

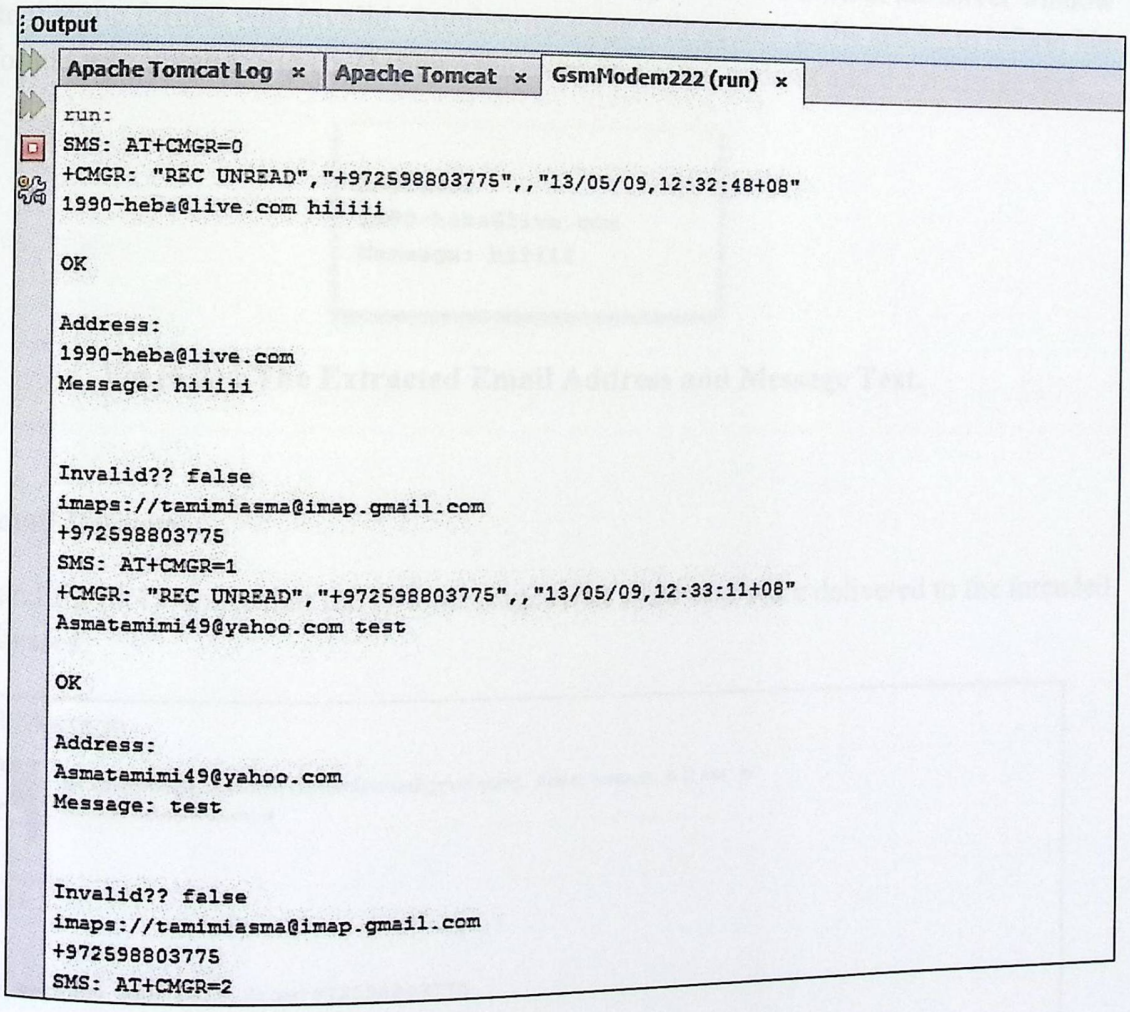


Fig (6.1): Process of Reading the Received Messages at the Server

The above figure shows that the server read two messages. As we can see, the details of the sent messages are shown. The details include:

1. The status of the message if it was read or not , is known through "REC UNREAD" or "REC READ".

2. The number of mobile from which the message was sent. In the above figure, mobile phone number was "+972598803775".

3. The date and Time of receiving the message through GPRS modem.

4. The sent message text.

The server then checked the format of the message. If the format was valid, the email address and the message body will be extracted from the original message. After that , the message will be sent to the intended email address. Otherwise, a message will be shown at the server window to indicate that the format was invalid. Another message will be sent to the sender to inform him that the format was invalid. Fig (6.2) shows the extracted email address and the message text.

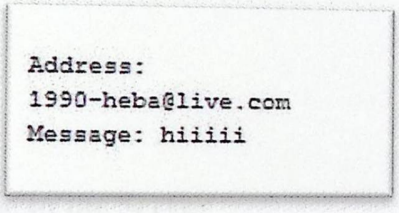


Fig (6.2) : The Extracted Email Address and Message Text.

6.2.1.2 Email Delivery

Fig (6.3) and fig (6.4) show that the two messages that were sent were delivered to the intended email addresses.

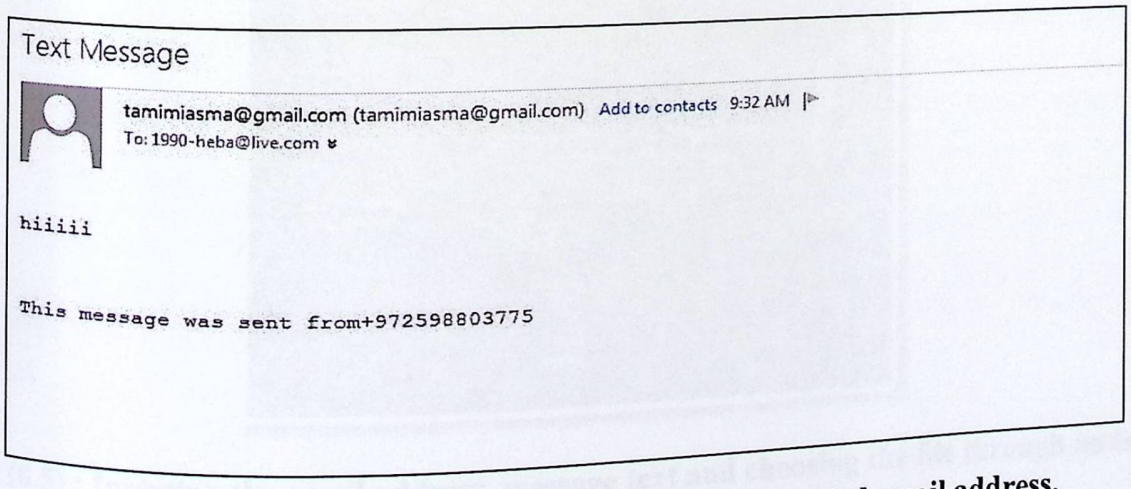


Fig (6.3): The first message was delivered to the intended email address.

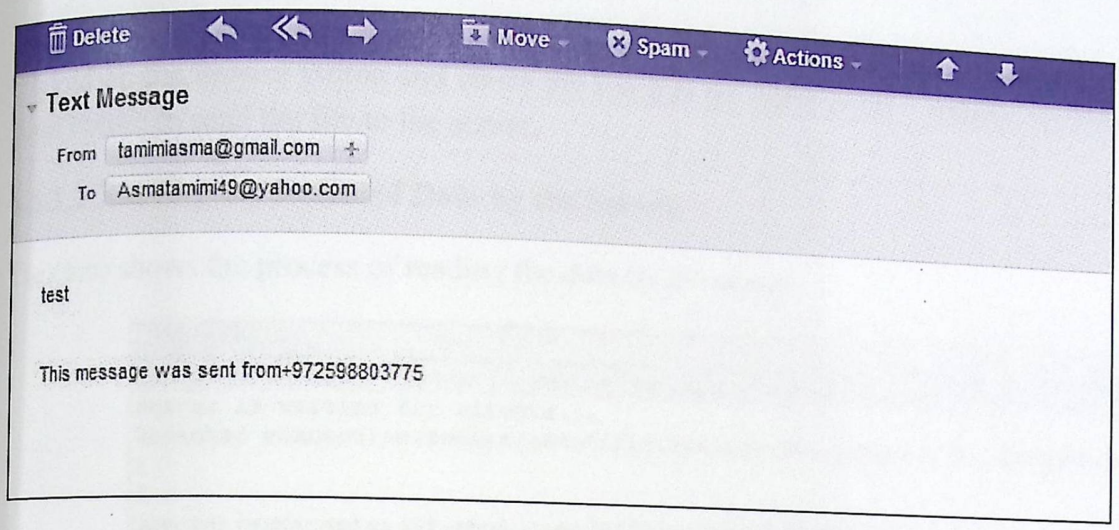


Fig (6.4): The second message was delivered to the intended email address.

6.3 File to Email Service

6.3.1Testing Phase

6.3.1.1 Sending File from Mobile to the Server

Fig (6.5) shows the process of inserting the email address, message text , and choosing the file.

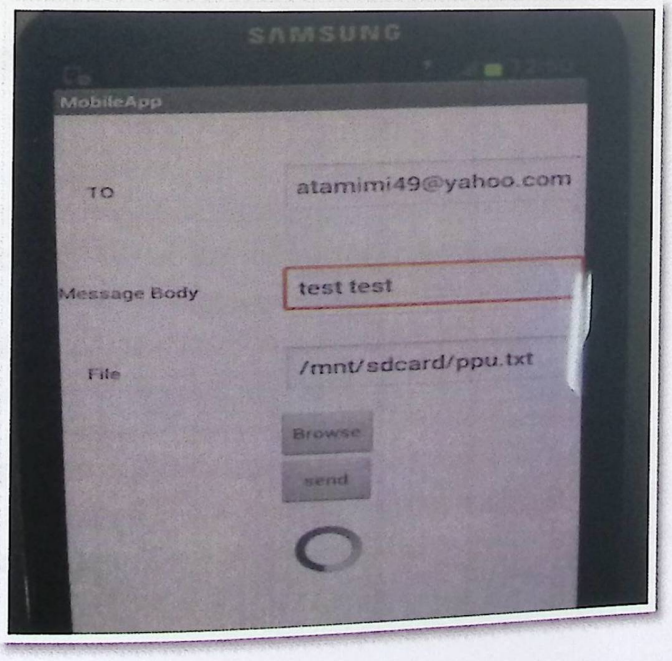


Fig (6.5) : Inserting the email address, message text and choosing the file through android application.

We inserted the email address which is a valid email address and the text message. After that, we clicked on the browse button and chose the file that we want to send. Finally, we clicked on the send button to send the file to the server.

6.3.1.2 Reading the Received Data by the Server.

Fig (6.6) shows the process of reading the data by the server

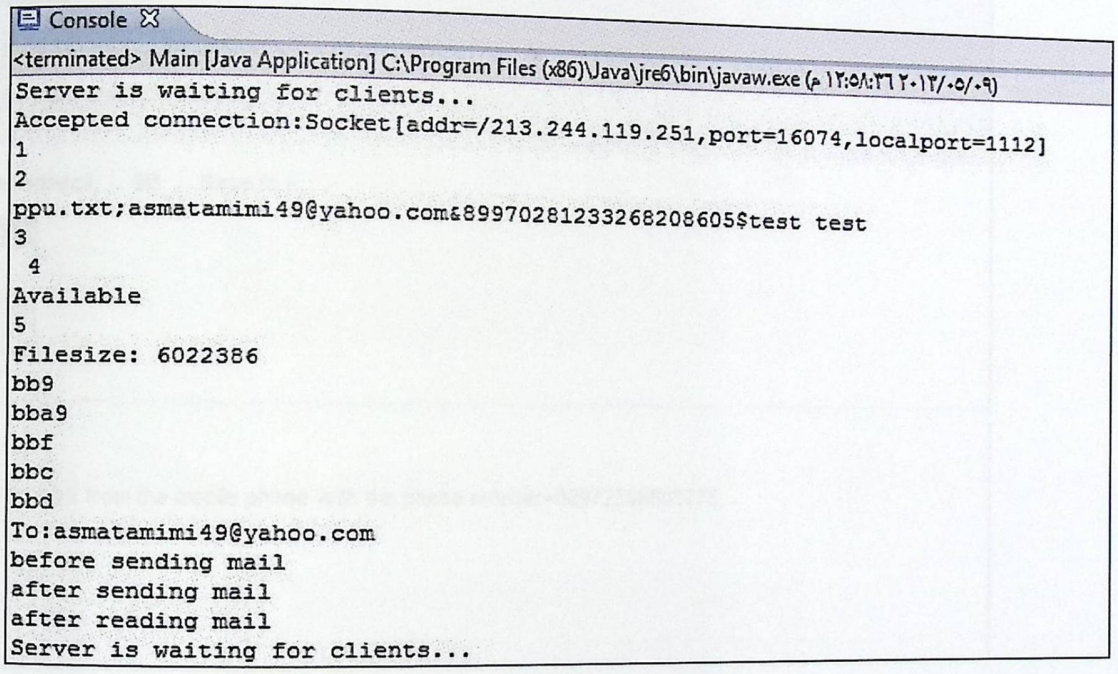


Fig (6.6): Reading the Received Data by the Server

First sentence *"Server is waiting for clients……"* shows that the server is ready for accepting any connection from any client. When the mobile connected to the server, the server accepted the connection as it is shown in the second sentence *"Accepted connection:Socket[addr=/213.244.119.251,port=16074,localport=1112]"*.It shows also the IP address and the port number of the mobile phone and the local port of the server.

The third sentence ""ppu.txt; asmatamimi49@yahoo.com &8997028123326820 8605 $test test" shows the name of the file, email address, serial number of the mobile and message text. After that, the server starts reading the file byte by byte. The file then will be sent to the intended email address.

6.3.1.3 Email Delivery.

Fig (6.7) shows that the text message and the file that we sent were delivered to the intended email address .

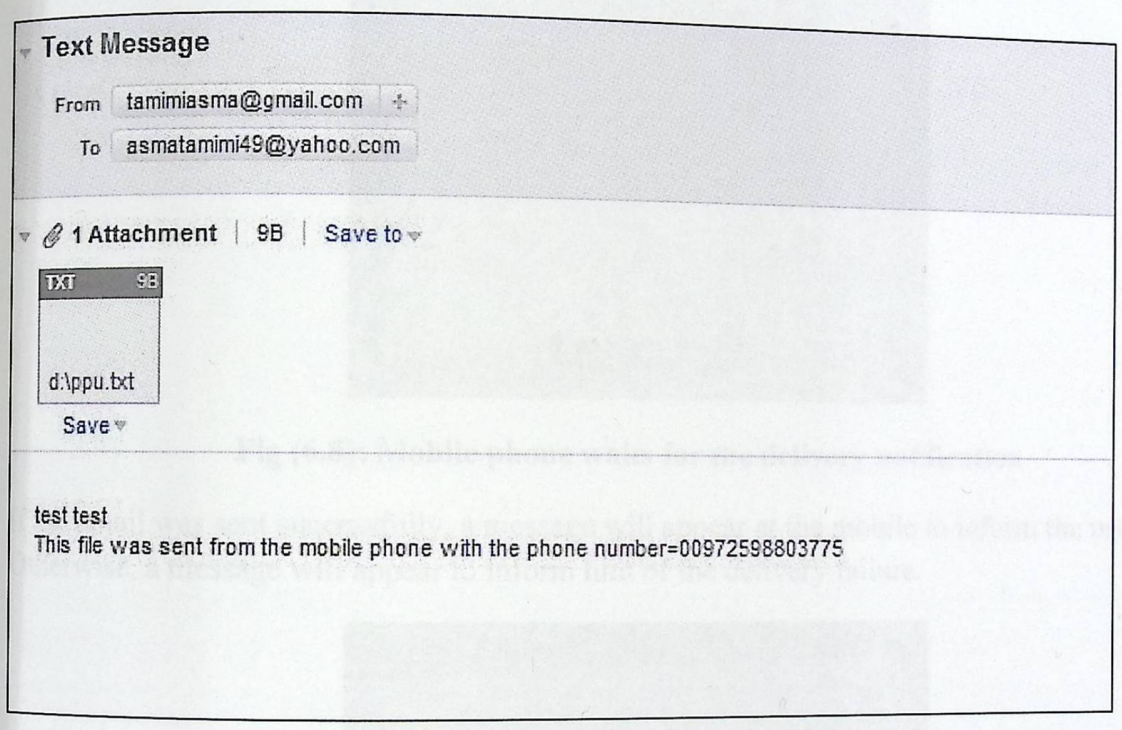


Fig (6.7): The file was delivered to the intended email address.

6.3.1.4 Receiving Delivery Notification

After sending the file, the mobile will wait until the notification arrives to inform him of the success or failure of the email delivery. Fig (6.8) shows the message that appears at the mobile to inform the user to wait until the notification arrives.

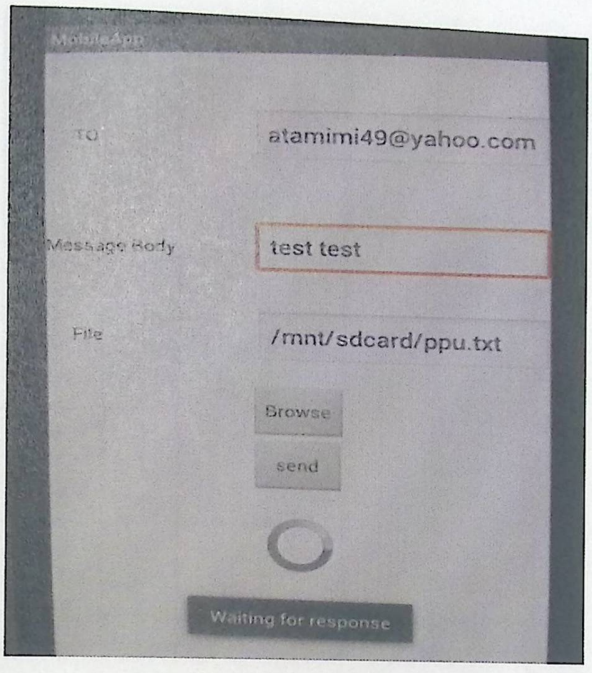


Fig (6.8): Mobile phone waits for the delivery notification

If the email was sent successfully, a message will appear at the mobile to inform the user of that. Otherwise, a message will appear to inform him of the delivery failure.

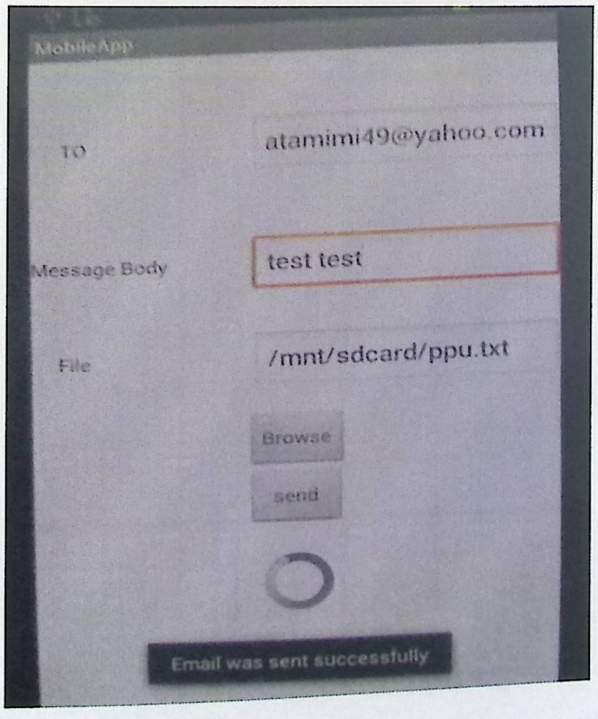


Fig (6.9): Mobile phone receives a notification of the delivery success.

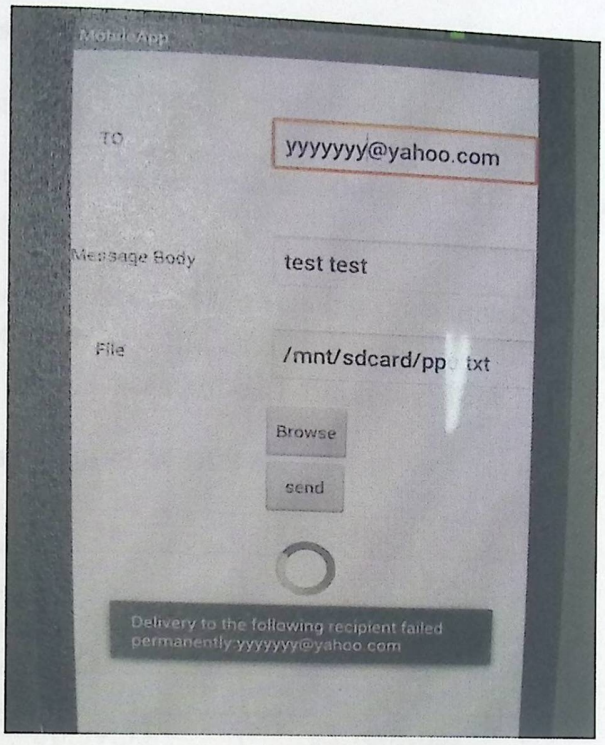
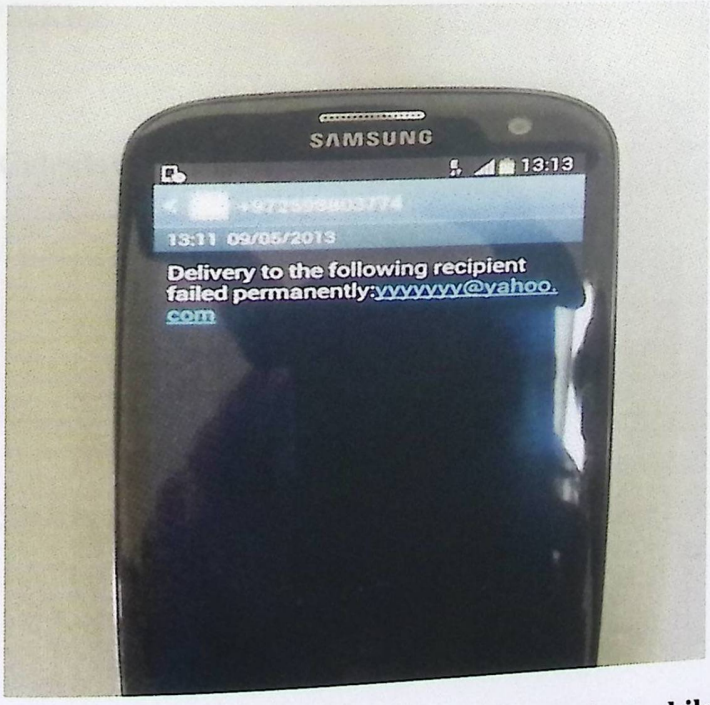


Fig (6.10): Mobile phone receives a notification of the delivery failure.

An SMS also was sent to the mobile phone to inform the user of the delivery failure as shown in fig (6.11).



Fig(6.11): SMS of the delivery failure was sent to the mobile phone.

6.4 Performance Analysis

6.4.1 Speed Calculations

Many types of files with different sizes were sent through file to email service. We sent text, images, pdf and mp3 files.

To calculate the time and the speed of the sending process using GPRS network, we sent two files. One of them was 463KB and the other file was 1.31 MB several times. Table (6.1) and table (6.2) show the time and the speed of each file at each time.

Table (6.1): Speed Calculation of 463KB file

Trial number	Time	Calculated speed
1	00:52:11	71.080Kbps
2	00:54:72	67.690Kbps
3	1:04:74	57.213Kbps
4	1:12:26	51.259Kbps
5	00:39:53	93.700Kbps
6	1:10:42	52.598Kbps
7	0:34:11	108.589Kbps
8	0:40:21	92.116Kbps

Average speed= 74.280Kbps

Table (6.2) : Speed Calculation of 1.31 MB file.

Trial number	Time	Calculated speed
1	1:19:00	132.658Kbps
2	1:07:86	154.435Kbps
3	1:07:19	155.975Kbps
4	1:13:72	142.159Kbps

Average speed=146.306Kbps

6.4.2 Success and Failure Rates

6.4.2.1 SMS to Email Service

Table (6.3) shows the number of successes and failures of each attempt.

Table (6.3): Number of Successes and Failures of each Attempt for SMS to Email Service

Trial number	Number of sent messages	Successes	Failures
1	3	2	1
2	5	4	1
3	9	9	0
4	3	2	1
5	6	6	0
6	7	6	1

Success rate=(2+4+9+2+6+6)/(3+5+9+3+6+7)=87.87%

Failure rate=(1+1+0+1+0+1)/(3+5+9+3+6+7)=12.12%

6.4.2.2 File to Email Service

We tried to send a file many times .Table (6.4) shows the number of successes and failures of each attempt.

Table (6.4): Number of Successes and Failures of each Attempt for File to Email Service

Trial number	File size	Attempts	Successes	Failures
1	1KB	11times	10	1
2	1.34MB	3times	3	0
3	1.57MB	4times	4	0
4	2.57MB	5times	5	0
5	7MB	6times	5	1
6	15.4MB	3times	2	1
7	24.7MB	2times	2	0

Success rate = (10+3+4+5+5+2+2) / (11+3+4+5+6+3+2) = 91.17 %

Failure rate = (1+0+0+0+1+1+0) / (11+3+4+5+6+3+2) = 8.82%

Chapter 7

Conclusion and Future Work

7.1 Introduction.

7.2 Conclusion.

7.3 Future Work.

7.1 Introduction

This chapter includes the conclusion of the project and the future work.

7.2 Conclusion

After 8 months of working on the project, the project team has learnt many things about android programming, java programming, AT commands , and configuration of GPRS modem. Moreover, they have learnt many things about writing the project documentation . Beside the technical side, the project team has gained a great experience in business side . They learnt how to write a business plan, business model, financial plan and design business cards .

The project team has achieved all the project goals and designed a reliable and robust system that enables the user to send emails from any place .

7.3 Future Work

As a next step, for those who are interested in developing this project or who want a new ideas to start working on it. We suggest the following ideas:

1) Design a system that enables the user to send MMS to email.

2) Notify the users of the incoming emails by sending SMS to their mobiles. By this service , the user will be able to know if a new email was delivered to his account.

Appendix A: SMS to Email Service Code

```
public static void main(String[] args) throws Exception {
        openPort();
        Timer t =  new Timer();
        TimerTask ReadMessageTask= new TimerTask(){@Override

        public void run() {
                 try {
                   String result = readSMSByIndex(index);
                   if(!(result.equals(""))){
                      if( !(result.indexOf("+CMS ERROR")>-1)){
                        if ((result.length()>70)){
                              System.out.println("SMS: "+result);
                              index++;
                              moNumb = result.substring(33, 46);

                              formatCheck(result.substring(71,
                              result.length()-4), moNumb);
                              System.out.println(moNumb);
                         }else{
                        }
                        }
                        }

                    } catch (Exception ex) {
                        System.out.println("ERROR: "+ex.getMessage());

                        Logger.getLogger(Main.class.getName()).log(Level.SE
                        VERE, null, ex);
                    }

                    }};

                     t.schedule(ReadMessageTask,0, 5000);
                       }


      public  static void openPort(){
          try{
              boolean boolPortOk=false;
              strPortName="com13";
              portList=CommPortIdentifier.getPortIdentifiers();
              while(portList.hasMoreElements()){
                 portId=(CommPortIdentifier)portList.nextElement();

                 if(portId.getPortType()==CommPortIdentifier.PORT_SERIAL){
                    if(portId.getName().equalsIgnoreCase(strPortName)){

                       Main.serialPort=(SerialPort)portId.open("SimpleWrit
                       eApp", 2000);
                       outputStream=serialPort.getOutputStream();
                       inputStream=serialPort.getInputStream();
                       serialPort.notifyOnDataAvailable(true);
```

```
                serialPort.setSerialPortParams(9600,
                SerialPort.DATABITS_8, SerialPort.STOPBITS_1,
                SerialPort.PARITY_NONE);
                boolPortOk=true;

                 break;
                 }
                 }
                 }
          if(!boolPortOk)throw new PortNotReadyException(strPortName);

      }  catch(Exception e){}
      }
 public static void closePort(){
       Main.serialPort.close();
       }
 public static  String readSMSByIndex(int index){
    try{
     final String INIT_ME="AT+CPMS=\"ME\"\r";
     final String TXT_MODE="AT+CMGF=1\r";
     final String AT_List="AT+CMGR="+index+"\r";
     StringBuffer sb=new StringBuffer();
     writeATCmd(INIT_ME);
     Thread.sleep(500);
     writeATCmd(TXT_MODE);
     Thread.sleep(500);
     sb.append(writeATCmd(AT_List));
     Thread.sleep(500);
     return sb.toString();
    }
    catch(Exception ex){
        System.out.print(ex.getMessage());
        return "err";
    }

}
private static String writeATCmd(String strATCmd){
     try{

     outputStream.write(strATCmd.getBytes());
     outputStream.flush();

     byte[]data=new byte[1024];
     int ch =inputStream.read(data);
     String str=new String(data,0,ch);
     return str;
     }
     catch(Exception i){
         return " 12";
     }
     }
```

```
final public static void sendSMS(String strPhNo,String strMsg)throws
Exception{
      final String STR_AT_CMGF="AT+CMGF=1\r";
      final String STR_AT_CMGS="AT+CMGS=\""+strPhNo+"\"\r";
      final String STR_CTRL_Z=strMsg+(char)26+"\r";

      try{
            writeATCmdSMS(STR_AT_CMGF);
            Thread.sleep(500);
            writeATCmdSMS(STR_AT_CMGS);
            Thread.sleep(500);
            writeATCmdSMS(STR_CTRL_Z);
            Thread.sleep(500);
            }catch(Exception e){
             System.err.println("/////Exception a:"
                        + e.getMessage());
            }

            }

  private static void writeATCmdSMS(String strATCmd)throws Exception{
       outputStream.write(strATCmd.getBytes());
       outputStream.flush();
       byte[] data=new byte[20];
       int ch=inputStream.read(data);

            }


  public static  void formatCheck(String str, String mobileNumber){
    boolean invalidFormatFlag= false;
       String mailAddress="", msg="";
       try{
           int firstSpacePosition;
           firstSpacePosition = str.indexOf(" ");
           if (firstSpacePosition<0){
               invalidFormatFlag = true;
                System.out.println("Invalid Format space");
           }else{
               mailAddress  = str.substring(0, firstSpacePosition);
               if(mailAddress.indexOf('@')<1){
                   invalidFormatFlag = true;
                   System.out.println("Invalid Format @");
               }else{

                 String
                 s=mailAddress.substring((mailAddress.indexOf('@')),
                 mailAddress.length());
                    if(s.indexOf('.')<(s.indexOf('@')+3) ||
                    mailAddress.indexOf('.')>= mailAddress.length()-
                    2){
                      invalidFormatFlag = true;
                      System.out.println("Invalid Format dot");
```

```
        }

        else{
            msg = str.substring(firstSpacePosition+1);

            System.out.println("Address: " + mailAddress);
            System.out.println("Message: " + msg);

        }
            }
        }

        System.out.println("Invalid?? " + invalidFormatFlag);
if(!invalidFormatFlag){
    String from= "tamimiasma@gmail.com";
    String to = mailAddress;
    String subject= "Text Message";
    String message= msg+"\n"+"This message was sent from"+
     mobileNumber;
    sendmail = new SendMail(from,to,subject,message);
    sendmail.send();
    Thread.sleep(3000);

     }
else{
    String msg0="Invalid message format";
    sendSMS(moNumb,msg0);
    }
    Properties props=System.getProperties();
    props.setProperty("mail.store.protocol", "imaps");

    try{
    Session session=Session.getDefaultInstance(props,null);
    Store store=session.getStore("imaps");

    store.connect("imap.gmail.com",sendmail.getUserID(),sendmail
    .getPassword());
    System.out.println(store);
    Folder inbox=store.getFolder("Inbox");
    inbox.open(Folder.READ_WRITE);
    FlagTerm ft=new FlagTerm(new Flags(Flags.Flag.SEEN),false);
    Message messages[]=inbox.search(ft);
    for(Message m:messages){
        if(m.getFrom()[0].toString().equals("Mail Delivery
         Subsystem <mailer-daemon@googlemail.com>")){
         String content = m.getContent().toString();
         System.out.println("Content >> " + content);
         to2=content.substring(64, content.indexOf(".")+4);
         System.out.println(to2);
          msg2="Delivery to the following recipient failed
          permanently:"+to2;

         sendSMS(moNumb,msg2);
         m.setFlag(Flags.Flag.DELETED, true);
```

```
                    }
                    }
                    inbox.close(true);
                    store.close();

                }
            catch(MessagingException e){
                    e.printStackTrace();
                        System.out.println("abc");
                    System.exit(1);

            }
            catch(Exception e){
                    e.printStackTrace();
                    System.exit(2);
            }

        }
     catch(Exception e){
    }

     }
}
```

Appendix B: File to Email Service(Client Code)

```
public class MainActivity extends Activity {
            static Timer getFailuresFrequentlyTimer,
            failuresFrequentlyTimerDurationTimer;
            TimerTask task;
            String IP="195.3.160.16";
            String TOMCAT_PORT="8092";
            String fileName="";
            String fileName2="";
            String message="";
            TextView v1,v2;
            static String destinationEmailAddress="";
            public static String path =null;
            public static EditText fN,msg;
            String SN;
            @Override
            protected void onCreate(Bundle savedInstanceState) {
                  super.onCreate(savedInstanceState);
                  setContentView(R.layout.activity_main);
                  SN=getSN();
                  setLabels();

            }

            private void setLabels() {

                  fN=(EditText)findViewById(R.id.fileText);
                  v2=(TextView)findViewById(R.id.FileLabel);
                  v2.setText("File");
                  v1=(TextView)findViewById(R.id.toLabel);
                  v1.setText("TO");
                  final Button b1;
                  Button b2;
                  b1=(Button)findViewById(R.id.button1);
                  b1.setText("Browse");
                  b2=(Button)findViewById(R.id.button2);
                  b2.setText("send");
            }

            private void checkFailuresTimerTask()
            {
                 this.runOnUiThread(generate2);
            }
            private Runnable generate2= new Runnable() {

              @Override
              public void run() {
                  getFailures();
             }

             };
            private void getFailuresFrequently() {
                  if(getFailuresFrequentlyTimer!= null){
```

```
            getFailuresFrequentlyTimer.cancel();
            getFailuresFrequentlyTimer = null;
        }
        getFailuresFrequentlyTimer = new Timer();

        getFailuresFrequentlyTimer.scheduleAtFixedRate(new
        TimerTask() {

            @Override
            public void run() {
                getFailuresFrequentlyTimerTask();
            }
        }, 2000,2000);
        }
private void setFailuresFrequentlyTimerDuration() {
      if(failuresFrequentlyTimerDurationTimer!= null){
            failuresFrequentlyTimerDurationTimer.cancel();
            failuresFrequentlyTimerDurationTimer = null;
      }
      failuresFrequentlyTimerDurationTimer = new Timer();
       failuresFrequentlyTimerDurationTimer.schedule(new
       TimerTask() {

            @Override
            public void run() {
                failuresFrequentlyTimerDurationTimerTask();
            }
        }, 40000);
    }
 private void failuresFrequentlyTimerDurationTimerTask()
    {
    this.runOnUiThread(generate3);
    }
    private Runnable generate3= new Runnable() {

        @Override
        public void run() {

            if(getFailuresFrequentlyTimer!=null){
              Toast.makeText(getApplicationContext(), "Email was
                   sent successfully", Toast.LENGTH_LONG).show();
              getFailuresFrequentlyTimer.cancel();
              getFailuresFrequentlyTimer= null;
            }

            }

            };
    private void getFailuresFrequentlyTimerTask()
    {
    this.runOnUiThread(generate);
    }
    private Runnable generate= new Runnable() {
```

```
        @Override
        public void run() {

            getFailures();


            }

            };

private void getFailures() {
      HttpURLConnection con=null;
      Toast makeText = Toast.makeText(this,"Waiting for
      response", Toast.LENGTH_LONG);
       try{
         makeText.show();
         URL url=new
         URL("http://"+IP+":"+TOMCAT_PORT+"/WebApplication1/NewSe
         rvlet1?sn="+SN);
         con = (HttpURLConnection) url.openConnection();
         String destinationAddress= readStream
         (con.getInputStream());
          if(!(destinationAddress.equals(""))){

              Toast.makeText(getApplicationContext(), "Delivery
              to the following recipient failed
              permanently:"+destinationAddress,
              Toast.LENGTH_LONG).show();

             if(getFailuresFrequentlyTimer!=null){

                getFailuresFrequentlyTimer.cancel();
                 getFailuresFrequentlyTimer=null;
                          }
                makeText.cancel();

                  }
       } catch (Exception e) {

           Toast.makeText(getApplicationContext(),
          e.getMessage(), Toast.LENGTH_LONG).show();
          e.printStackTrace();
            }
       finally{
            if(con!=null)
                  con.disconnect();
       }
}
```

```
public void enableGprs(){
            try{
                final ConnectivityManager conma(ConnectivityManager)
                getApplicationContext().getSystemService(Context.CONNECT
                IVITY_SERVICE);
                final Class conmanClass =
                Class.forName(conman.getClass().getName());
                final java.lang.reflect.Field iConnectivityManagerField
                = conmanClass.getDeclaredField("mService");
                iConnectivityManagerField.setAccessible(true);
                final Object iConnectivityManager =
                iConnectivityManagerField.get(conman);
                final Class iConnectivityManagerClass =
                Class.forName(iConnectivityManager.getClass().getName())
                ;
                final Method setMobileDataEnabledMethod =
                iConnectivityManagerClass.getDeclaredMethod("setMobileDa
                taEnabled", Boolean.TYPE);
                setMobileDataEnabledMethod.setAccessible(true);


                setMobileDataEnabledMethod.invoke(iConnectivityManager,
                true);
            }
            catch(Exception n){}
            }
private boolean isGPRSEnabled(){
            final ConnectivityManager mngr;
            mngr= (ConnectivityManager)this.getSystemService
            (Context.CONNECTIVITY_SERVICE);

             final NetworkInfo gprs;
             gprs=mngr.getNetworkInfo(ConnectivityManager.TYPE_MOBIL
             E);
             return gprs.isConnected();
        }


public void sendFileToServer(View v){
    Socket sock;
    PrintWriter out;
try{
     if(!isGPRSEnabled()){
         Toast.makeText(this.getApplicationContext(), "Connecting GPRS...
         ",Toast.LENGTH_LONG).show();
         Thread.sleep(2000);
         enableGprs();
                  }
    else
         Toast.makeText(this.getApplicationContext(), "GPRS Is already
         Enabled: ", Toast.LENGTH_SHORT).show();

    while(!isGPRSEnabled()){
         Thread.sleep(500);
```

```
                }

                sock=new Socket(IP,1112);
                EditText emailAddressView;
                emailAddressView=(EditText)findViewById(R.id.toText);

                destinationEmailAddress=emailAddressView.getText().toString();
                msg=(EditText)findViewById(R.id.editText1);
                message=msg.getText().toString();
                out= new PrintWriter(new BufferedWriter(new
                OutputStreamWriter(sock.getOutputStream())),true);

                out.println(fileName2+";"+destinationEmailAddress+"&"+SN+"$"+messa
                ge);
                File myFile=new File(fileName);
                byte [] mybytearray= new byte[(int)myFile.length()];
                FileInputStream fis = new FileInputStream(myFile);
                BufferedInputStream bis = new BufferedInputStream(fis);
                bis.read(mybytearray, 0, mybytearray.length);
                OutputStream os= sock.getOutputStream();
                Toast.makeText(this.getApplicationContext(), "Sending.....",
                Toast.LENGTH_SHORT).show();
                os.write(mybytearray, 0, mybytearray.length);
                os.flush();
                sock.close();
                sock= null;
                getFailuresFrequently();
                setFailuresFrequentlyTimerDuration();
                }catch(UnknownHostException e){
                Toast.makeText(this.getApplicationContext(), e.getMessage(),
                Toast.LENGTH_SHORT).show();
                e.printStackTrace();

                 }
          catch(IOException e){
                Toast.makeText(this.getApplicationContext(), e.getMessage(),
                Toast.LENGTH_SHORT).show();
                e.printStackTrace();
                        }
        catch(Exception e){
                Toast.makeText(this.getApplicationContext(), e.getMessage(),
                Toast.LENGTH_SHORT).show();
                e.printStackTrace();
                        }
                }

        public void browse(View v){
                  Intent i=new Intent(this,ListV.class);
                  startActivityForResult(i,0);
                  }
        public void onActivityResult(int requestCode,int resultCode,Intent data){
                  fN=(EditText)findViewById(R.id.fileText);
                  fileName=data.getExtras().getString("Path");
                  fileName2=data.getExtras().getString("fileName");
```

```
        fN.setText(fileName);
 }
 private String readStream(InputStream in) {
        String line = "22222",m="";
   BufferedReader reader = null;
   try {
     reader = new BufferedReader(new InputStreamReader(in));

     while ((line = reader.readLine()) != null) {
       m=m+line;
     }
   } catch (IOException e) {
         line=e.getMessage();
     e.printStackTrace();
   } finally {
     if (reader != null) {
       try {
         reader.close();
       } catch (IOException e) {
         e.printStackTrace();
         }
     }
   }
   return m;
}
public  String getSN()
{
TelephonyManager mTelephonyMgr = (TelephonyManager)
this.getSystemService(Context.TELEPHONY_SERVICE);
String sn = mTelephonyMgr.getSimSerialNumber();
  if( sn == null )
        {
  return "";
        }
 Toast.makeText(this.getApplicationContext(), sn,
 Toast.LENGTH_SHORT).show();

        return sn;

}
      @Override
      public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if   it
        is present.
            getMenuInflater().inflate(R.menu.activity_main, menu);
            return true;
      }

}
```

Appendix C: File to Email Service(Server Code)

```
public class ServerClass extends Thread {
      int filesize = 6022386;
      int count ;
      Socket sock;
      Timer t;
      Timer t2;
      BufferedReader in;
      int port;

 public ServerClass(int port) {
            this.port = port;
      }

      public void run() {
        try {
          ServerSocket servsock = new ServerSocket(port);
          while (true) {
          System.out.println("Server is waiting for clients...");

          do {
            sock = servsock.accept();

              } while (sock == null);

                    in = new BufferedReader(new InputStreamReader(
                    sock.getInputStream()));
                    System.out.println("Accepted connection:" + sock);

                    t = new Timer();
                    t2=new Timer();

                    String to = "";
                    String to1 = in.readLine();
                    String f = to1.substring(0, to1.indexOf(";"));
                    String msg2;
                    to = to1.substring(to1.indexOf(";") + 1, to1.indexOf("&"));
                    String sn = to1.substring(to1.indexOf("&") + 1,
                    to1.indexOf("$"));
                    msg2 = to1.substring(to1.indexOf("$") + 1);

                    /////Selecting the phone number from the database
                    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                    String dataSourceName = "database";
                    String dbURL = "Jdbc:odbc:" + dataSourceName;
                    Connection con = DriverManager.getConnection(dbURL, "",
                    "");
                    Statement s = (Statement) con.createStatement();
                    s.execute("select PhoneNo from Table1 where SN='" + sn +
                    "'");
                    ResultSet b = s.getResultSet();
                    if (b.next())
```

lix C: File to Email

```
class ServerClass
int filesize = 6022386;
int count ;
Socket sock;
Timer t;
Timer t2;
BufferedReader in;
int port;

ac ServerClass(int port) {
      this.port = port;
}

public void run() {
  try {
    ServerSocket servsock = new ServerSocket(
    while (true) {
    System.out.println("Server is
    do {
      sock = servsock.accept();

        } while (sock == null);

          in = new BufferedReader(new InputStreamReader(
          sock.getInputStream()));
          System.out.println("Accepted connection:" + sock);

          t = new Timer();
          t2=new Timer();

          String to = "";
          String to1 = in.readLine();
          String f = to1.substring(0, to1.indexOf(","));
          String msg2;
          to = to1.substring(to1.indexOf(",") + 1, to1.indexOf("%"));
          String er = to1.substring(to1.indexOf("%") + 1,
          to1.indexOf("$"));
```

```
String phoneNo = b.getString("PhoneNo");
con.close();
///// Receiving the File
byte[] mybytearray = new byte[filesize];
InputStream is = sock.getInputStream();
File tempFile = new File("d:\\" + f);
if (tempFile.exists())
    tempFile.delete();
    FileOutputStream fos = new FileOutputStream("d:\\" +
    f);
    BufferedOutputStream bos = new
    BufferedOutputStream(fos);

    while ((count = is.read(mybytearray)) > 0)
    {
          System.out.println("bba" + count);
          bos.write(mybytearray, 0, count);
    }

    bos.flush();
    bos.close();
    fos.close();

    //// Sending email
    System.out.println("To:" + to);
    String from = "tamimiasma@gmail.com";
    String subject = "Message";
    String message = msg2;
    String str = "This file was sent from the mobile
    phone with the phone number=";
    System.out.println("before sending mail");
    SendMail1 sendmail = new SendMail1(from, to, subject,
    message, f, str + phoneNo);
    sendmail.send();
    System.out.println("after sending mail");

    ////Reading delivery failure emails
    t.schedule(new TimerTask() {

          @Override
    public void run() {
    readEmail();
              }
    }, 0, 7000);
    System.out.println("after reading mail");
    t2.schedule(new TimerTask(){

          public void run(){
                stopTimer();
          }


          }, 30000);
          }
```

```
} catch (Exception e) {
      System.out.println(e.getClass().toString());
      System.out.println(e.getMessage().toString());
}

finally {
try {
       JOptionPane.showMessageDialog(null, "Closed");
       if (in != null)
             in.close();

       if(sock != null)
       sock.close();

} catch (IOException e) {
       // TODO Auto-generated catch block
       e.printStackTrace();
}
}

}

   private void stopTimer() {
   if(t!=null){
   t.cancel();
   t=null;
     }
     }

 public void readEmail() {
 String PNo, to2;
 try {
Properties props = System.getProperties();
props.setProperty("mail.store.protocol", "imaps");
Session session = Session.getDefaultInstance(props, null);
Store store = session.getStore("imaps");
store.connect("imap.gmail.com", "tamimiasma", "******");
System.out.println(store);
Folder inbox = store.getFolder("Inbox");
inbox.open(Folder.READ_WRITE);
FlagTerm ft = new FlagTerm(new Flags(Flags.Flag.SEEN),
false);
Message messages[] = inbox.search(ft);
for (Message m : messages) {
     if (m.getFrom()[0].toString().equals("Mail Delivery
     Subsystem <mailer-daemon@googlemail.com>")) {
      String content = m.getContent().toString();
      System.out.println("Content >> " + content);

     /////Extracting the email address and phone number from
     the content of the delivery failure email
```

```
                to2 = content.substring(64,   content.indexOf(".") +
                 4);
                System.out.println(to2);
               PNo = content.substring(content.indexOf("mobile phone
                with the phone number=") + 35);
                PNo = PNo.trim();
                System.out.println(PNo);


               /////Updating the database by inserting the extracted
               destination

                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                String dataSourceName = "database";
                String dbURL = "Jdbc:odbc:" + dataSourceName;
               Connection con =   DriverManager.getConnection(dbURL,
               "", "");
                Statement s = (Statement) con.createStatement();
                s.execute("UPDATE Table1 SET Destination = '" + to2
                            + "' where  PhoneNo='" + PNo +    "'");
                con.commit();
                con.close();

                 /////Sending SMS of the delivery failure to the user
                 SendSMS sendSMS1 = new SendSMS();
                 sendSMS1.openPort();
                 String msgerr = "Delivery to the following recipient
                 failed permanently:"+ to2;
                 sendSMS1.sendSMS(PNo, msgerr);
                 sendSMS1.closePort();
                 m.setFlag(Flags.Flag.DELETED, true);
                 t.cancel();
                 t = null;
                 }
                 }
                 inbox.close(true);
                 store.close();

} catch (MessagingException e) {
      e.printStackTrace();
      System.exit(1);

} catch (Exception e) {
      System.out.println(e.getMessage());
}

}
}
```

Appendix D: Sending Email Code

```
public class SendMail1 {
        private String from;
        private String to,f,d;
        private String subject;
        private String text;

public SendMail1(String from, String to, String subject, String text,String
           f,String d){
            this.from = from;
            this.to= to;
            this.subject=subject;
            this.text=text;
            this.f=f;
            this.d=d;

        }

        public void send(){
            String host="smtp.gmail.com";
            String userid="tamimiasma";
            String password="******** ";

            try
            {
                Properties props= System.getProperties();
                props.put("mail.smtp.starttls.enable","true");
                props.put("mail.smtp.host", host);
                props.setProperty("mail.transport.protocol", "smtps");
                props.put("mail.smtp.user", userid);
                props.put("mail.smtp.password", password);
                props.put("mail.smtp.port", "465");
                props.put("mail.smtps.auth","true");
                Session session = Session.getDefaultInstance(props, null);
                MimeMessage message= new MimeMessage(session);
                InternetAddress fromAddress = null;
                InternetAddress toAddress= null;

                try{
                    fromAddress= new InternetAddress(from);
                    toAddress= new InternetAddress(to);
                }
                catch(AddressException e)
                {
                    e.printStackTrace();
                }
                message.setFrom(fromAddress);
                message.setRecipient(RecipientType.TO, toAddress);
                message.setSubject(subject);
                //message.setText(text);

                //=========================================
```

Appendix D: Sending Email Code

```
public class SendMail1 {
        private String from;
        private String to,f,d;
        private String subject;
        private String text;

public SendMail1(String from, String to, String subject, String text,String
          f,String d){
           this.from = from;
           this.to= to;
           this.subject=subject;
           this.text=text;
           this.f=f;
           this.d=d;

       }

       public void send(){
           String host="smtp.gmail.com";
           String userid="tamimiasma";
           String password="******** ";

           try
           {
               Properties props= System.getProperties();
               props.put("mail.smtp.starttls.enable","true");
               props.put("mail.smtp.host", host);
               props.setProperty("mail.transport.protocol", "smtps");
               props.put("mail.smtp.user", userid);
               props.put("mail.smtp.password", password);
               props.put("mail.smtp.port", "465");
               props.put("mail.smtps.auth","true");
               Session session = Session.getDefaultInstance(props, null);
               MimeMessage message= new MimeMessage(session);
               InternetAddress fromAddress = null;
               InternetAddress toAddress= null;

               try{
                   fromAddress= new InternetAddress(from);
                   toAddress= new InternetAddress(to);
               }
               catch(AddressException e)
               {
                   e.printStackTrace();
               }
               message.setFrom(fromAddress);
               message.setRecipient(RecipientType.TO, toAddress);
               message.setSubject(subject);
               //message.setText(text);

               //=========================================
```

Appendix D: Sending Email Code

```
public class SendMail1 {
        private String from;
        private String to,f,d;
        private String subject;
        private String text;

public SendMail1(String from, String to, String subject, String text
        f,String d){
         this.from = from;
         this.to= to;
         this.subject=subject;
         this.text=text;
         this.f=f;
         this.d=d;

    }

    public void send
        String host="
        String user=
        String pass

        try
        {
            Properties
            props.put
            props.put
            props.set
            props.put
            props.put
            props.put
            props.put
            props.put
            Session sess
            MimeMessage
            InternetAdd
            InternetAdd

            try{
                fromAddress
                toAddress=

            catch(AddressExc
            {
                .printStackTr

            .setFrom(fr
            .setRecipien
            .setSubject(
            .setText(te
```

```
            // Create the message part
                BodyPart messageBodyPart = new MimeBodyPart();

                // Fill the message
                messageBodyPart.setText(text+"\n"+d);

                Multipart multipart = new MimeMultipart();
                multipart.addBodyPart(messageBodyPart);

                // Part two is attachment
                messageBodyPart = new MimeBodyPart();
                String filename = "d:\\"+f;
                DataSource source = new FileDataSource(filename);
                messageBodyPart.setDataHandler(new DataHandler(source));
                messageBodyPart.setFileName(filename);
                multipart.addBodyPart(messageBodyPart);

                // Put parts in message
                message.setContent(multipart);


              //=========================================

              Transport transport = session.getTransport("smtps");
              transport.connect(host, userid, password);

              transport.sendMessage(message, message.getAllRecipients());
              transport.close();
            }
            catch(MessagingException e){
                e.printStackTrace();
            }

        }

}
```

References:

- [1] *Short Message Service/SMS Tutorial* .Available at: http://www.developershome.com/sms/
- [2] NOR ALIA BINTI HASSAN, "SMART HOME SYSTEM BY SENDING SMS VIA GSM MODEM", Faculty of Electronic and Computer Engineering ,Universiti Teknikal Malaysia Melaka, APRIL 2011.
- [3] Maher Abdel-qader, "Using Short Message Service (SMS) to Support Business Continuity", Department of Computer Science, Amman Arab University for Graduate Studies , World of Computer Science and Information Technology Journal (WCSIT) ,Vol. 1, No. 2, 34-38, 2011.
- [4] Malik Sikandar Hayat Khiyal, Aihab Khan, and Erum Shehzadi ,"SMS Based Wireless Home Appliance Control System(HACS) for Automating Appliances and Security",Software Engineering Dept., Fatima Jinnah Women University,Rawalpindi, Pakistan, Issues in Informing Science and Information Technology, Volume 6, 2009.
- [5] Muhammad Samiullah, Noman Sohaib Qureshi, Hira Nazir," SMS Repository and Control System using GSM-SMS Technology", Department of Computer Science & Engineering University of Engineering and Technology Lahore, European Journal of Scientific Research,Vol.70 No.4 (2012), pp. 499-507.
- [6] *GSM Tutorial*. Available at: http://ecee.colorado.edu/~ecen4242/gsm/index.htm.
- [7] Rahul Gondane, *Mobile Generations-1G to 4G* .March 16,2011 .Available at: *http://www.coolpctips.com/2011/03/mobile-generations-1g-to-4g/*
- [8] *Cellular networks generation of standards for mobiles phones, telecommunications, data services (0G, 1G, 2G, 3G, 4G, 5G)*. Available at: *http://cwh0623.blogspot.com/*
- [9] Thomas Farely,*GSM History* .Jan 15,2006 . Available at : http://www.privateline.com/mt_gsmhistory/02_gsm_history/
- [10] Gwenae¨l Le Bodic, "Mobile Messaging Technologies and Services: SMS, EMS and MMS" , JOHN WILEY & SONS, LTD,2003

- [11]*GSM OVERVIEW*. Available at: *http://gsmoverview.blogspot.com/*
- [12] *GSM: Network Architecture*. Available at: http://www0.cs.ucl.ac.uk/staff/t.pagtzis/wireless/gsm/arch.html
- [13] Fraser Sherman ,*The History of GPRS*. Available at : *http://www.ehow.com/info_12154824_history gprs.html#ixzz2FBlF7OsE*
- [14] Sabeer Bhatia and others, *GPRS Architecture* .Available at : http://www.tutorialspoint.com/gprs/gprs_architecture.htm.
- [15]*GPRS Tutorial*, Available at: http://www.tutorialspoint.com/gprs/gprs_architecture.htm
- [16] *The history of SMS Messaging*. Available at: http://www.avtext.com/the-history-of-sms-messaging.php
- [17] Veena K.Katankar & Dr.V.M.Thakare, "Short Message Service using SMS Gateway", International Journal on Computer Science and Engineering, page 1487,2010
- [18] Puneet Gupta,*Short Message Service: What, How and Where?*.Available at: http://www.wirelessdevnet.com/channels/sms/features/sms.html
- [19] Gwenael Le Bodic, Mobile Messaging Technologies and services SMS, EMS and MMS, book, page 44-46, JOHN WILEY & SONS, LTD
- [20]http://www.developershome.com/sms/sms_tutorial.asp?page=egApps
- [21] Now.sms, *What is a GSM Modem*. Available at:http://www.nowsms.com/faq/what-is-a-gsm-modem
- [22]OZEKI NG SMS Gateway ,*GSM Modem* . available at : http://ozekisms.com/index.php?owpn=543
- [23] WiseGEEK, *What is a GSM Modem?*. Available at: http://www.wisegeek.com/what-is-a-gsm-modem.htm
- [24] SMS solutions.net, *Overview of GSM/GPRS Modems.* Available at: http://www.smssolutions.net/hardware/gsmgprs1/
- [25] EngineersGarage , AT Commands, *GSM AT command set* .Available at : .http://www.developershome.com/sms/atCommandsIntro.asp

- [26] *A brief history of email*. Available at: http://www.vicomsoft.com/learning-center/history-of-email/

- [27] *GSM: Radio interface*.Available at: http://www0.cs.ucl.ac.uk/staff/t.pagtzis/wireless/gsm/radio.html