

***J2ME based E-mail client
for Java-enabled mobile devices***

by
Anthony J Solon (98607693)
BSc (Hons) Computing Science
Project Report, April 26, 2002

Abstract

The retrieval of e-mail using a wireless device will become widespread with the next phase of mobile devices, especially with the arrival of the third generation (3G) of mobile networks and the vast improvements in data transmission speeds that will accompany it.

This project is concerned with the provision of a POP3 (Post Office Protocol version 3) based e-mail application that will be used to provide remote mailbox access to users of Java-enabled mobile phones or Personal Digital Assistants (PDAs). The application will be developed using the J2ME (Java 2 Micro Edition) toolkit. For e-mail sending and retrieval functionality the POP3 and SMTP protocols will be used to retrieve and send email messages respectively.

As the project is concerned with the development of an e-mail application destined for use on mobile devices I have named the application Me-Mail, which stands for Mobile e-mail.

Acknowledgements

I would like to extend my sincere thanks and appreciation to my project supervisor, Mr. Kevin Curran, for his guidance and advice. I also extend my sincere thanks and appreciation to my family, whose support throughout the past years of my college education only made it all possible.

Contents

Abstract.....	2
Acknowledgements.....	3
Contents	4
Table of Figures	7
Chapter 1 – INTRODUCTION	8
1.1 Background.....	8
1.2 Objectives	9
1.3 Thesis Outline.....	9
1.4 Criteria for Success.....	10
Chapter 2 – LITERATURE SURVEY	11
2.1 Mobile Communication.....	11
2.1.1 “1G” wireless technology.....	11
2.1.2 “2G” wireless technology.....	11
2.1.3 “2.5G” wireless technology.....	12
2.1.4 “3G” wireless technology.....	13
2.2 Java 2 Micro Edition	14
2.2.1 Introduction.....	14
2.2.2 J2ME Configuration.....	16
2.2.3 J2ME Profile	17
2.2.4 CLDC	17
2.2.5 MIDP.....	18
2.2.5.1 MIDlets	18
2.2.5.2 MIDP User Interfaces	19
2.2.5.2 Network Programming in J2ME MIDP	20
2.2.6 CDC.....	20
2.2.7 Foundation Profile.....	21
2.2.8 KVM – CVM	22
2.2.9 Advantages of J2ME	23
2.2.10 Disadvantages/Limitations of J2ME	24

2.2.11 J2ME Prospects:	24
2.3 E-mail System	26
2.3.1 SMTP	27
2.3.2 MIME	28
2.3.3 E-mail Retrieval	29
2.3.4 Offline	29
2.3.5 Online	29
2.3.6 Disconnected	30
2.3.7 POP	30
2.3.8 IMAP	31
2.4 JavaMail API	31
2.5 Other Technologies	32
2.5.1 WAP	32
2.5.2 i-Mode	33
2.5.3 WAP & i-Mode Limitations	33
2.5.4 SMS	34
Chapter 3 – REQUIREMENTS & DESIGN	35
3.1 Introduction	35
3.2 Functional Requirements	36
3.3 Non-Functional Requirements	38
3.4 Hardware & Software Requirements	39
3.4.1 Hardware	39
3.4.2 Software	40
3.5 Methodology Overview	40
3.5.1 Chosen Methodology	41
3.6 User Interface	43
3.7 Design	45
3.8 Summary	50
Chapter 4 – IMPLEMENTATION	51
4.1 Introduction	51
4.2 Implementation	52

4.3 SMTP Implementation	54
4.4 POP3 Implementation	56
4.5 User Interface Design	60
4.6 Iterative Development	62
4.7 Implementation summary	62
Chapter 5 – EVALUATION AND TESTING	63
5.1 Introduction	63
5.2 Evaluation	63
5.2.1 Functional Requirements Evaluation	65
5.2.2 Non-Functional Requirements Evaluation	69
5.2.3 User Interface Tests	71
5.3 Testing	73
5.4 Summary	77
Chapter 6 - CONCLUSION	79
6.1 Future Work	80
References	82
<i>Books:</i>	82
<i>Web sites:</i>	84
Appendix A – Connection.java	87
Appendix B – J2meConnectioImpl.java	87
Appendix C – SmtplibClient.java	88
Appendix D – Message.java	89
Appendix E – Envelope.java	90
Appendix F – SmtplibException.java	91
Appendix G – Pop3Client.java	91
Appendix H – InboxClient.java	92
Appendix I – Pop3Exception.java	92
Appendix J – MimeDecoder.java	92
Appendix K– MeMail.java	93
Appendix L – User.java	97

Table of Figures

Figure 1: Architecture of the Java Technology.....	15
Figure 2: Architecture of the various virtual machines, configurations and profiles.	22
Figure 3: Examples of phones providing MIDP Java™ support.	25
Figure 4: Interaction between two SMTP servers.....	27
Figure 5: Structure of an e-mail message	28
Figure 6: Flow chart for Me-mail.	45
Figure 7: High-level use case diagram & subsequent elementary use case diagrams.	47
Figure 8: Class diagram of the Me-Mail system.....	48
Figure 9: Architecture of network connection.	54

Chapter 1 – Introduction

1.1 Background

With the advent of 3G networks and subsequent increased speed in data transfer available, the possibilities for applications and services that will link people throughout the world who are connected to the network will be unprecedented.

Among these applications will no doubt be many new innovative ideas, but the application considered by this project is alien to very few people, due to the ubiquitous use of computers and electronic mail in today's society.

I am basing this project on the e-mail system due to the fact that it has already claimed its place in the communication channels for both corporate and personal use. It provides us with our primary means of sending and receiving information electronically to and from one another. In fact a report published by IDC [IDC, 2001] forecasts e-mail usage between now and 2005. Included in the report is the conservative prediction that the number of e-mail mailboxes in use worldwide is going to more than double.

Hence the necessity for an implementation of the e-mail service on mobile devices would seem evident, as mobile networks evolve into a more powerful medium for data transmission and the devices that are connected to them inevitably grow in their ability to process the data at increasingly faster rates. The evolution of these devices is backed by Moore's law which states that processor power will almost double every 18 months.

One may even anticipate a time when the applications available on wireless devices will replace the original versions implemented on ordinary desktop computers. Although this situation is far from realisation taking into consideration the current state of the technology available for accessing 3G networks or even 2.5G networks.

1.2 Objectives

To research, develop and implement a solution for sending and retrieving e-mail messages to and from a Java-enabled mobile phone, implementing SMTP for sending messages and POP3 for the retrieval functionality.

1.3 Thesis Outline

The first chapter of this thesis gives a comprehensive description of the main areas of technology that have been investigated during the literature review phase of this project. The areas examined are various technologies that are related to the Me-mail framework. The commencing section, 2.1, introduces the various different types of mobile networks that have evolved to date.

Once these Generations of mobile technologies have been explained the following section 2.2 introduces the architecture of the Java technology. It identifies the three editions of the Java programming language that exists, paying particular attention to the Java 2 Micro Edition and the various standards that comprise it. This section also includes the advantages and limitations of Java 2 Micro Edition, and concludes by discussing the prospects for Java 2 Micro Edition in the future.

Section 2.3 delves into the fundamentals of E-mail systems by explaining and comparing the underlying protocols involved in sending and receiving e-mail messages.

Following on from this, in section 2.4 the JavaMail API is explained. This API provides a convenient method for implementing the protocols described in the previous section into an e-mail client application designed using the Java language. Section 2.5 is devoted to discussing technologies that are used on mobile devices to provide access to information and services.

Chapter three deals with the requirements and design of the application. Functional, non-functional, software and hardware requirements are defined. Methodologies are discussed briefly followed by an explanation of the design.

Chapter four then outlines the implementation phase of the project and details the classes involved in development. This chapter is broken into: SMTP implementation, POP3 implementation, User Interface design, and Iterative development.

Chapter five describes the Evaluation and testing processes. Functional and non-functional requirements evaluation results are documented. Then User Interface testing and evaluation is described. To conclude this chapter, a complete test-run through the system is illustrated.

Chapter six includes the conclusion and a section detailing potential future work if the project was continued.

1.4 Criteria for Success

At this point it is important to set a criteria for success – this will provide a reference point when evaluation and testing are under way.

Therefore, this project will have succeeded when it has fulfilled the following criteria:

The system accesses both POP3 and SMTP servers and implements the necessary functionality associated with each. A UI has been developed for the application which displays Inbox contents or message contents on the client's screen in a logical, well structured fashion.

Chapter 2 – Literature Survey

2.1 Mobile Communication

Mobile phone technologies have evolved in several major phases denoted by “Generations” or “G” for short. Three generations of mobile phones have evolved so far, each successive generation more reliable and flexible than the previous.

2.1.1 “1G” wireless technology

The first of these is referred to as the first generation or 1G. This generation was developed during the 1980s and early 1990s. It only provided an Analog voice service with no data services available.

2.1.2 “2G” wireless technology

The second generation or 2G of mobile technologies used circuit-based, digital networks. Since 2G networks are digital they are capable of carrying data transmissions, with an average speed of around 9.6K bps (bits per second). The three standards involved in 2G networks are Time Division Multiple Access (TDMA), Code Division Multiple Access (CDMA) in America and Global System for Mobile (GSM) used in both America and Europe.

“GSM has by far the largest number of users on a worldwide scale (600 million and growing as of fall 2001, approximately two thirds of all cellular users in the world today).” [Pressman, 1999]

Because 2G networks can support the transfer of data, they are able to support J2ME enabled phones. Some manufacturers are providing J2ME phones for 2G networks though the majority are designing their Java enabled phones for the 2.5G and 3G networks, where the increased bandwidth and data transmission speed will make these applications more usable.

2.1.3 “2.5G” wireless technology

“2.5” is an acronym which represents various technology upgrades to the existing 2G mobile networks. Upgrades to increase the number of consumers the network can service while boosting data rates to around 56K bps. 2.5G upgrade technologies are designed to be overlaid on top of 2G networks with minimal additional infrastructure. Examples of these technologies include: General Packet Radio Service (GPRS) and Enhanced Data rates for Global Evolution (EDGE). They are packet based and allow for “always on” connectivity.

Introducing 2.5G makes it possible for today’s mobile operators to gain vital business and market experience by providing high speed mobile data services based on packet switching. This also has the advantage that it demonstrates to mobile users the possibilities of 3G.

GPRS was introduced in 2000 in France, but due to handset shortage and technical problems with the overall network architecture it was not a success.

2.1.4 “3G” wireless technology

Following on from analog and digital technology, the third generation of mobile communications will be digital mobile multimedia offering broadband mobile communications with voice, video, graphics, audio and other forms of information.

3G builds upon the knowledge and experience derived from the preceding generations of mobile communication, namely 2G and 2.5G. Although, 3G networks use different transmission frequencies from these previous generations and therefore require a different infrastructure.

Because of this, operators and carriers are spending billions in building out their 3G network infrastructures. Most networks are not expected to be completely built out until at least 2003-2005. These networks will improve data transmission speed up to 144K bps in a high speed moving environment, 384K bps in a low-speed moving environment, and 2M bps in a stationary environment.

3G services see the logical convergence of two of the biggest technology trends of recent times, the Internet and mobile telephony. Some of the services that will be enabled by the broadband bandwidth of the 3G networks include:

- Downloadable and streaming **Audio** and **Video**.
- Voice Over Internet Protocol (**VoIP**).
- Send/receive **Still Images**.
- Send/receive **Moving Images** – allows for video-conferencing, also downloadable movies.
- Virtual Home Environments (**VHE**).
- **Electronic Agents** – “self contained programs that roam communications networks delivering and receiving messages or looking for information or services”. [3G Newsroom, 2000]
- **Downloadable Software** – This is more environmentally friendly (no packaging), quicker and more convenient than conventional methods of distributing software as the product arrives in minutes.

- Capability to determine geographic position of a mobile device – Global Positioning System (**GPS**).

3G will also facilitate many other new services that have not previously been available over mobile networks due to the limitations in data transmission speeds.

These new wireless applications will provide solutions to companies with distributed workforces, where employees need access to a wide range of information and services via their corporate intranets, when they are working offsite with no access to a desktop.

The many services 3G will make available are expected to not only result in a corporate revolution but a consumer revolution also. No matter which side one looks at, the users involved will demand more interactive and personalised applications that can significantly improve their lifestyles and greatly simplify the way they conduct business. J2ME appears to be the best available solution at present to meet these challenges.

2.2 Java 2 Micro Edition

2.2.1 Introduction

Sun Microsystems defines Java 2 Micro Edition (J2ME) as “A highly optimised Java run-time environment targeting a wide range of consumer products, including pagers, cellular phones, screen-phones, digital set-top boxes and car navigational systems”. [Feng et al. 2001]

Below is a diagram showing what devices and platforms each of the three editions of the Java technology are designed to provide solutions for. Java 2 Enterprise Edition (J2EE) provides solutions for the enterprise environment, Java 2 Standard Edition

(J2SE) for desktop development and low-end business applications, and J2ME for consumer and embedded devices.



Figure 1: Architecture of the Java Technology

[EuropeMedia, 2001]

Java dates back to 1991, when researchers (Patrick Naughton, Mike Sheridan and James Gosling) at Sun Microsystems were attempting to create a programming language for use in consumer devices. However, the breakthrough for Java as a mainstream computer language came in 1995 when Netscape made the next version of their browser Java-enabled. Java and the Java VM together provide a set of services that Java programs can rely on, regardless of the underlying hardware and operating system.

Announced in June 1999, J2ME brings the cross-platform functionality of the Java language to smaller devices, allowing mobile wireless devices to share applications. There is no J2ME specification due to the fact that it isn't a single standard. It is, instead, a group of related specifications that define what Java technology looks like on resource-constrained devices.

Before I discuss and explain the various specifications that J2ME consists of, it's important to appreciate the broad variety of devices that could implement J2ME, ranging from mobile phones to entertainment systems. From this appreciation it is easily recognisable that it would be impossible to define a single technology that would be optimal, or even close to optimal for all these devices. The differences between them as regards processor power, memory, persistent storage and user interface are simply too severe. To address this problem, Sun divided and then sub-divided the definition of devices suitable for J2ME into sections.

The first division separated devices based on processing power, memory, and storage capability. This resulted in the creation of two J2ME configurations: Connected, Limited Device Configuration (CLDC) and Connected Device Configuration (CDC).

2.2.2 J2ME Configuration

Feng et al. 2001 define a configuration as:

“A J2ME Configuration defines class libraries for a category or grouping of devices based on similar requirements for a total memory budget and processing power.”

Because there is such variability across user interface, function, and usage even within a configuration, these areas are not defined within a typical configuration. The definition of that functionality belongs, instead, to what is called a Profile.

2.2.3 J2ME Profile

Feng et al. 2001 states that a profile is “Built on top of a specific configuration, a J2ME profile defines class libraries to address the specific demands of a certain market segment.”

In other words a profile addresses a specific class of device, such as pagers and mobile phones. It supplements the already existing functionality in a configuration for these devices by sitting on top of the configuration. There are currently two profiles: the Foundation Profile (FP) which supplements the CDC, and the Mobile Information Device Profile (MIDP) which supplements the CLDC.

2.2.4 CLDC

CLDC is the smaller of the two J2ME configurations, aimed at serving devices that have the following characteristics:

- 16-bit or 32-bit processor.
- Limited memory – 512 KB or less.
- A limited power supply (typically a battery).
- Limited or intermittent network connectivity.
- A simple user interface, if any at all.
- Limited screen size.
- Limited input abilities

CLDC is concerned with applications destined for use on mobile phones, pagers, Personal Digital Assistants (PDAs), and similar devices. In order for the CLDC to fit onto these devices with such memory limitations much of the standard functionality present in the Java 2 Standard Edition (J2SE) is abandoned. For example, the `java.util.package`, which contains forty-seven classes and interfaces in the J2SE, was

reduced to only ten classes in the CLDC. The result is that CLDC does not provide all of the functionality required to build useful applications (E.g. user interface classes). But this was Sun's intention, as CLDC was not intended to be a complete solution; it is a common base upon which profiles targeted at specific product types can add functionality to.

2.2.5 MIDP

The Mobile Information Device Profile (MIDP) adds to the CLDC the functionality necessary to build useful applications for mobile phones, two-way pagers and similar devices. MIDP re-implements functionality to support user interfaces, timers, simple persistent storage, networking and messaging. Applications developed to the MIDP specification are called Midlets.

2.2.5.1 MIDlets

An application developed for use on a MIDP device is referred to as a MIDlet. It does not have a main method; instead, a MIDlet extends the 'javax.microedition.midlet.MIDlet' class and implements its three abstract methods: startApp(), pauseApp(), and destroyApp(). A MIDlet must also define a public no-argument constructor which is the method instantiated by the Application Management Software (AMS) on the MIDP device when a MIDlet is launched. In order to develop an application to the MIDP/CLDC specifications, the MIDP and CLDC class packages must be downloaded [JavaSun J2ME, 2001] and their locations added to the classpath in the IDE where the MIDlet is being developed.

All the class files belonging to a MIDlet are packaged into a single JAR file. This can then be downloaded and installed onto a wireless device via a serial cable connected

to a PC or over-the-air (OTA) via a wireless network. This JAR file must also include a *manifest* file. A manifest file contains attributes describing the contents of a JAR such as name, version, and vendor of the MIDlet. One other file which the JAR must include is an *application descriptor* which is used in the deployment process of a MIDlet (especially OTA). Before a JAR is downloaded to a device, the application descriptor is checked by the AMS to ensure the MIDlet is suited to the device.

One other point to be made about packaging a MIDlet is that the code must be preverified in addition to compiling. Traditionally in J2SE, the verification process is performed at runtime by the Java Virtual machine (JVM). But due to the resource constraints on wireless devices, class verification is performed partially off-device and partially on-device to improve run-time performance. The off-device verification is referred to as *preverification*. The preverifier inserts a stack map attributes into the class file to help the in-device verifier quickly scan through the byte codes without costly operations.

To aid with the packaging of MIDlets Sun provide the J2ME Wireless Toolkit (J2MEWTK) [JavaSun J2ME, 2001]. With this toolkit one can streamline several of the necessary tasks (creation of manifest file and application descriptor, compilation using MIDP compiler, and preverification) into one easy step. This facility is provided by a part of the toolkit called the Ktoolbar. Alternatively the J2MEWTK can be integrated with the ‘Forte for Java’ IDE. These features are subsequently accessible via ‘Forte for Java’ instead.

2.2.5.2 MIDP User Interfaces

The User Interface defined in MIDP is logically composed of two sets of APIs:

- **High-level UI API** which emphasises portability across different devices
- **Low-level UI API** which emphasises flexibility and control.

The portability in the **high-level API** is achieved by employing a high level of abstraction. The actual drawing and processing user interactions are performed by implementations. Applications that use the high-level API have little control over the visual appearance of components, and can only access high-level UI events.

On the other hand, using the **low-level API**, an application has full control of appearance, and can directly access input devices and handle primitive events generated by user interaction. However the low-level API may be device-dependent, so applications developed using it will not be portable to other devices with a varying screen size.

2.2.5.2 Network Programming in J2ME MIDP

J2ME replaces the networking packages in J2SE with the Generic Connection Framework. To meet the small-footprint requirement necessary in mobile devices, the Generic Connection Framework generalises the functionality of J2SE's network and file I/O classes: 'java.io' and 'java.net'. It is a precise functional subset of the J2SE classes, but much smaller in size. The J2ME classes and interfaces are all included in a single package 'javax.microedition.io', which supports the following forms of communication: HTTP, Sockets, Datagrams, Serial Ports, and Files.

2.2.6 CDC

CDC is the larger of the two J2ME configurations (the other being the CLDC already described above). It is best used for developing applications for television set top boxes, entertainment systems, automobile navigation systems (GPS), home appliances, point-of-sale terminals, and other devices of that scale.

These devices generally have the following features:

- Are powered by a 32-bit processor.
- Have 2MB or more of memory.
- Are connected to a network, often a wireless connection which is inconsistent and has limited bandwidth.
- Are designed with user interfaces that have varying degrees of sophistication, or maybe have no interface at all.
- They can support a complete implementation of the standard Java Virtual Machine (JVM) and Java language.

The CDC JVM named the CVM must be compatible with the standard JVM and Java programming language (J2SE). CDC provides the minimal set of APIs required to support the standard JVM. The CDC configuration adds APIs for file input and output, networking, security, object serialisation, and a few more along with the ones already available in the CLDC.

But CDC like CLDC does not include any type of user interface classes, as user interfaces vary considerably from device to device.

2.2.7 Foundation Profile

Like MIDP is the profile that supplements the CLDC, similarly Foundation Profile (FP) is the profile defined for CDC. It extends considerably the APIs provided by CDC, however it does not provide the user interface APIs. FP must be augmented by one or more additional profiles that provide user interface support. Personal Profile is one such profile.

2.2.8 KVM – CVM

J2ME configurations also simplify the features of their underlying Java virtual machines compared with the standard JVM. CDC and CLDC each come with their own optimised virtual machine.

The underlying virtual machine for CLDC is the K Virtual Machine (KVM).

The KVM is a very small, yet very functional Java virtual machine specifically designed for resource-constrained devices. The K stands for Kilo as its memory is measured in kilobytes.

The virtual machine for CDC is the C Virtual Machine (CVM). CVM is a fully featured, small footprint version of the standard JVM used with the J2SE specifically designed for high-end consumer devices.

The figure below depicts the relationship between the different virtual machines, configurations and profiles. It also draws a parallel with the J2SE API and it's Java virtual machine. KVM and CVM can be thought of as shrunken versions of the J2SE JVM.

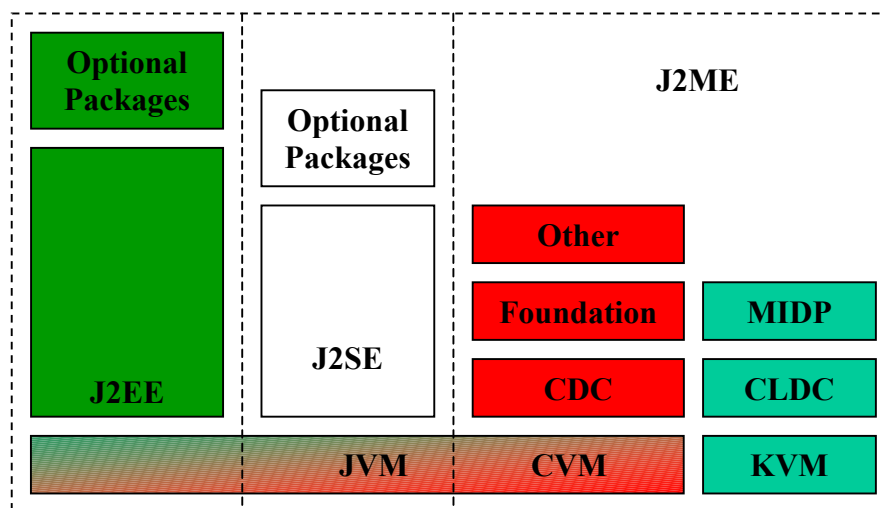


Figure 2: Architecture of the various virtual machines, configurations and profiles.

2.2.9 Advantages of J2ME

- **Platform Independence** – Java code is architecturally neutral and can be run on any variety of systems, as long as they possess a Java Interpreter. Wireless devices vary from each other in shape and functionality also, and this is why it's essential that J2ME extend the original "Write Once, Run Anywhere" design philosophy to the wireless world. Wireless applications developed using Java can be run on different devices from different vendors. This greatly improves the portability of a J2ME program.
- **Rich Network Functionality** – Wireless applications by nature are network oriented and provide the user with constant communication to the outside world anywhere, anytime. Java is designed with network capabilities in mind; it provides a rich set of network libraries.
- **Built in Security Model** – Java provides several levels of security, from class loader and bytecode verifier to Security Manager, which can protect client systems from untrustworthy programs. Java also provides extended security APIs for securely transforming content over the Web. This approach allows safer transactions for mobile commerce applications and financial applications.
- **Dynamic Application Deployment** – Most of the existing applications on wireless devices are built in, fixed-feature applications. It is very difficult to upgrade and install new applications without getting the manufacturer involved. J2ME provides a dynamic deployment mechanism that allows applications to be downloaded and installed onto devices over the wireless network. This dynamic application deployment not only provides a cost-effective way for vendors and developers to distribute software products, but it also allows users to download applications on demand and personalise their on-device applications dynamically.
- **Distributed Computing** – The strong XML support in Java makes client/server or transaction-based applications feasible on wireless devices.
- Java is wholly **Object-Oriented** – this brings the benefits of the Object-Oriented paradigm, such as inheritance and reuse.

- **Graphical User Interface** – Just like the user interface support in J2SE, J2ME comes with a rich set of user interface and event handling class libraries that make the most of the limited display space on wireless devices. This user interface support makes sophisticated games and complex entertainment applications feasible on wireless devices.
- **Developer Community** – “The good news for device manufacturers is that by opening their development platform to the Java community, they have gained access to 2.5 million Java talents.” [Feng et al. 2001]
- Other advantages of using Java in wireless devices include that Java overcomes low bandwidth hurdles; it alleviates memory constraints and it is scalable.

2.2.10 Disadvantages/Limitations of J2ME

- The KVM (virtual machine for CLDC) has no Java Native Interface (JNI); also it isn't possible to access Palm's infrared port or serial port.
- Also due to the memory restrictions on a small device, there is no way of implementing a Just-In-Time compiler (JIT). As a result the code runs about three to eight times slower than a native application written in a language such as C.
- Consideration must also be given to the fact that the virtual machine takes up valuable memory space. A native application would probably be around the size of the virtual machine alone.

2.2.11 J2ME Prospects:

Cahners In-Stat Group predicts that:

“The number of MID subscribers will grow to more than 1.3 billion in 2004 with sales of more than 1.5 billion wireless handsets, PDAs and Internet appliances.”

[JavaWorld, 2001]

A majority of companies are already choosing and supporting J2ME in order to provide a solution for their wireless applications. Companies that have worked on the MIDP include Ericsson, NEC, Nokia, NTT, DoCoMo, Palm Computing, Sansung, Sony, Alcatel, Psion, Siemens, Motorola and many more. Taking the lead in defining the MIDP is Motorola. The previous list of well-known mobile device manufacturers and mobile service providers shows that there is a significant vested interest in the J2ME technology. Considering this strong industry backing J2ME can really become the new platform to serve millions of mobile users with new, truly interactive, graphically appealing content.

“Java’s requirements exceed what today’s handsets deliver” [JavaWorld, 2001]

Hence, the wireless handsets available today need to evolve further before they will be capable of providing the services that become possible when using J2ME.

But as I mentioned earlier, there is a large alliance of companies working to improve this situation, by producing more powerful handsets. Some of these companies are designing Java enabled phones for use with the current second-generation (2G) of networks and the 2.5G of networks, although the vast majority are focusing their interests on phones that are usable on 3G networks. According to JavaMobiles, 2001 there are about 34 Java enabled phones available at present, provided by various manufacturers. Below are some examples of some java enables phones, Motorola Accompli 008, Nokia 7650 and Siemens SL45i:



Figure 3: Examples of phones providing MIDP Java™ support.
[JavaMobiles, 2001]

An example of such a company is Sprint PCS who says it'll use applications based on the Java language when it deploys it's third-generation (3G) of wireless networks in mid-2002. Sprint is the forth-largest wireless carrier in the US. They are supporting J2ME because of its support for more graphical and personalised interactive services, as well as games. Sprint said "The Java platform will enable consumers to continually upgrade applications on a device and to interoperate with a variety of devices because it is an industry standard". [JavaWorld, 2001]

2.3 E-mail System

E-mail (Electronic Mail) is the exchange of messages by telecommunications. Email has advantage over other types of sending messages or information in that it's cheaper and faster than a letter, less intrusive than a phone call, less hassle than a FAX and it isn't limited in length like the Short Message Service (SMS).

A report published by IDC [IDC, 2001] forecasts e-mail usage between now and 2005. Included in the report is the conservative prediction that the number of e-mail mailboxes in use worldwide is going to more than double.

The use of e-mail messaging in Ireland since 1997 has grown from 100,000 users to 1.25 million users as of July 2001. [IDC, 2001]

Most e-mail systems implement two basic protocols. One of these handles outgoing mail sent by the user and the other is concerned with incoming mail.

2.3.1 SMTP

The Simple Mail Transfer Protocol (SMTP) is the protocol that is used when a user sends an e-mail message. The mail client on the user's machine communicates with the SMTP server on port 25 in order to submit the message for delivery. The SMTP server on the user's host may also have to interact with other SMTP servers to actually the mail is the receiver is not local to that mail host. Figure 4 depicts this:

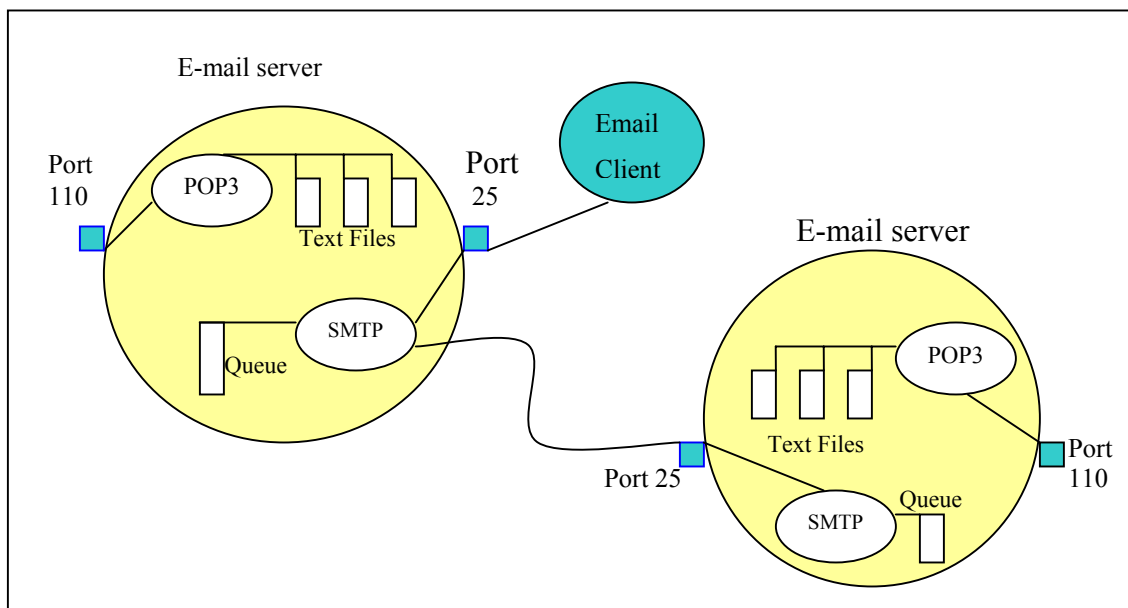


Figure 4: Interaction between two SMTP servers

SMTP can only handle messages containing 7-bit ASCII text. This means that SMTP is unable to handle types of data such as 8-bit binary data and other multimedia formats. Since many people want to attach additional files to their messages, and e-mail messages can only contain text information; there is a problem that needs to be solved.

2.3.2 MIME

MIME (Multi purpose Internet Mail Extensions) is not a transfer protocol like SMTP. Instead it defines the content of what is transferred: the format of the message, attachments and so on. This overcomes the limitations of SMTP in that it packs multimedia data into formats that SMTP can handle. MIME uses a program called uuencode which produces an encoded version of the original binary file (attachment) that only contains text characters. Modern e-mail clients run uuencode and uudecode (the program to translate the data back into a binary file) automatically for the user, therefore providing a way to send and receive e-mail messages with attachments of any type seamlessly. Below is a diagram showing the structure of an e-mail message, which clearly shows how the MIME part is integrated.

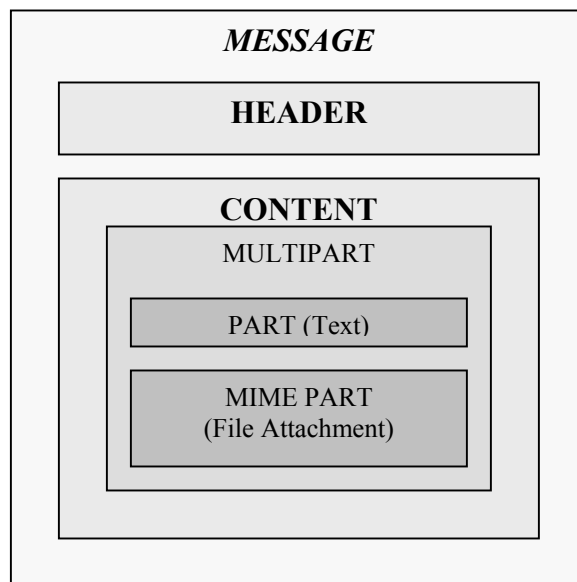


Figure 5: Structure of an e-mail message

2.3.3 E-mail Retrieval

There are three fundamental models for remote mailbox access: *offline*, *online* and *disconnected*. Each presenting a completely different method for retrieving e-mail from the mail server.

2.3.4 Offline

Offline access is the most familiar form of client/server e-mail today. In this model a client application periodically connects to the mail server and downloads any new messages to the client; removes them from the server; and then disconnects (hence. “Offline”). Subsequent mail processing (reading, deleting etc) is done locally on the client. This minimises use of server resources and should lead to a shorter connection-time, as the user doesn’t have to be online when he/she is just reading e-mails. This is important for dial-up users. A short connection time is not always the case though, as when one of the new messages has a large file attached, this will result in a long download time and a longer connection time, which is not at all desirable for a dialup user.

2.3.5 Online

In this model, a client application manipulates the mailbox directly on the server, maintaining a connection throughout the session. No mailbox data is stored locally, the client only retrieves data from the server as is needed. Therefore in this type of mailbox access if the user isn’t connected to a network, he/she cannot access their e-mail messages. This model places a much heavier burden on the server, requiring the

server to receive the new messages, deliver them to the user when requested and maintain them in multiple folders for each user.

2.3.6 Disconnected

Disconnected user access is a blend of the online and offline models. In this model the user connects to the server periodically, downloads a specified set of messages from the server, manipulates them offline, and later uploads the changes. The server remains the authoritative repository of the messages. Synchronisation is handled by assigning a unique identifier to each message.

2.3.7 POP

POP3 stands for Post Office Protocol version 3. This protocol describes a transfer method for retrieving e-mail from a POP3 server, which is listening on port 110. It was designed to support “offline ” mail processing. As described previously this means that the client downloads all new messages from the mail server including files attached to these messages using MIME. Then these messages are deleted from the server. POP servers can also be asked to retain messages after they have been downloaded. This mode of operation using POP is called “pseudo offline”. For online and disconnected access though, POP’s simplicity is a drawback.

2.3.8 IMAP

IMAP stands for Internet Messaging Access Protocol. It provides support for the three paradigms of remote mailbox access: online, disconnected and offline; online being the default model used. The protocol permits manipulation of mailboxes as if they were local. An IMAP client can ask the server for headers or the bodies of specified messages, or to search for messages meeting certain criteria. In comparison with POP3, IMAP offers the following main advantages:

- Richer functionality in manipulating an inbox.
- The ability to manage mail folders besides the inbox.
- More efficient methods for dealing with large MIME messages.

But IMAP also has its down sides; it requires more server storage and also it results in longer connection times as the user is manipulating data on the server.

2.4 JavaMail API

The JavaMail API released by JavaSoft is a collection of abstract classes that model a mail system. It provides the fundamental set of functions that any basic mail-handling system must support. The JavaMail API provides functionality for the following specific activities:

- Creating a mail message.
- Creating a session object. A session object validates a JavaMail user, and controls the user's access to the message storage and transport services.
- Sending a mail message.
- Retrieving a message. The API provides methods for searching the message repository on the mail server and retrieving messages from it.

The JavaMail API enables a user to build standards based e-mail clients in the Java language that employ various Internet mail protocols, including SMTP, POP3, IMAP

and MIME. Although, this method of e-mail client development is primarily aimed at standard desktop computer, as the resulting package is too large for use on wireless devices.

2.5 Other Technologies

WAP and i-Mode are the two dominant technologies that allow users to access the Internet from their mobile phones. WAP technology has been widely adopted by wireless carriers in Europe and the U.S., whereas i-Mode is very popular in Asia.

2.5.1 WAP

Phone.com (now OpenWave) a merged venture with Softwate.com, first introduced WAP (Wireless Application Protocol) in 1995. WAP is an open standard that enables easy delivery of information and services to mobile users. It was intended to address the need to access the Internet from handheld devices. WAP defines a set of standard components that enable the communication between mobile terminals and network servers, including the WAP programming model, WAP communication protocol, Wireless Markup Language (WML), WMLScript scripting language, and micro-browser. WAP has received wide support in the wireless industry. Currently, several million subscribers worldwide use WAP-enabled mobile phones to access the Internet.

WML is a markup language similar to HTML. WMLscript is a scripting language that extends the computational capabilities of WML. A micro-browser that resides on wireless devices can interpret WML and WMLScript and present content to users. People have been using WAP to shop, get weather information, trade stocks and more, all from their mobile phones.

2.5.2 i-Mode

The Japanese company NTT DoCoMo first introduced i-Mode. This technology competes with WAP as it offers a similar mechanism to allow users to access the Internet from their wireless devices over a packet-switched network.

Feng et al, 2001 estimates that as of December 2000, about 16 million people use i-Mode-enabled mobile phones to access the Internet. The majority of i-Mode users are in Japan and other Asian countries.

i-Mode comes with its own markup language, compact HTML (cHTML), which is a subset of ordinary HTML along with some of NTT's own tags or characters. Because cHTML is a subset of HTML, i-Mode pages can also be viewed by using a regular Web browser. i-Mode phones come with a mini-browser that allows the user to access Web content written with cHTML.

2.5.3 WAP & i-Mode Limitations

Both the WAP and i-Mode technologies are comparable to HTML and Web browsers for desktop computers. They provide the platform for delivering information content to wireless devices.

However, WAP and i-Mode are limited in some ways, including lack of security, poor user interfaces, and the requirement of constant airtime for standalone or offline operations, which is a huge disadvantage as connections to WAP services are charged at high rates with most mobile carriers. As the wireless industry matures, users demand more security for mobile commerce transactions, more interactive applications such as video games, and more sophisticated applications for client/server enterprise applications. Java 2 Micro Edition is designed to meet these challenges. Although it must be added that both MID for J2ME and WAP are

intended to be complimentary, as WAP is primarily a transport mechanism and J2ME is an execution piece.

2.5.4 SMS

Short Message Service (SMS) allows mobile phone users to send and receive short text messages. These e-mail type messages are of very limited length, allowing up to only 160 characters when Latin based alphabets are used, and 70 characters when non-Latin based alphabets such as Arabic and Chinese are used. SMS has several features that make it unique:

- It is reliable. Unlike pagers if the recipient doesn't have to have their device turned on or in coverage the message is lost, while SMS will store the message at the SMSC (SMS Center) and will send it again once they can be reached.
- SMS offers you the ability of receiving confirmation that a message has been delivered.
- SMS allows for simultaneous transmission with GSM voice, data and fax services.

Chapter 3 – Requirements & Design

3.1 Introduction

The requirements can be defined as “*A complete understanding of the software development effort. No matter how well designed or well coded, a poorly analysed and specified program will disappoint the user and bring grief to the developer.*” [Pressman, 1999]

Sommerville emphasises that the “*system requirements should set out **what** the system should do rather than **how** this is done. A requirement may be a **functional requirement**, that is, it describes a system service or function. Alternatively, it may be a **non-functional requirement**. A non-functional requirement is a constraint placed on the system (for instance, the required response time) or on the development process (such as the use of a specific language standard).*” [Sommerville, 1997]

“*The Design process involves developing several models of the system at different levels of abstraction. As a design is decomposed, errors and omissions in earlier stages are discovered.*” [Sommerville, 1997]

“*Design is a creative process. Although methods and guidelines are helpful, judgment and flair are still required to design a software system.*” [Sommerville, 1997]

This following section will discuss the system requirements i.e. ‘what the system should do rather than how this is done’ [Sommerville, 1995, p64]. Requirements may be either:

- **Functional**, describing the facilities of the system
- **Non-functional**, describing constraints which may impact later development phases

3.2 Functional Requirements

Functional requirements describe the facilities the system has to offer. Each description should use natural language without resorting to technical jargon, and simple diagrams so each requirement is understandable to customers, users and management. In this project, each requirement will be individually labeled to ensure tractability when testing commences, thus ensuring each requirement is tested.

The functional requirements for this system are sub-divided into two sections, as there are two distinguishable areas of functionality dealt with, namely Sending e-mail messages and Retrieving e-mail messages from the mail server.

The main areas of functionality include:

- 1.1 - Initially the user will be able to **Login**, where they have already added themselves as a user in the application and configured their user settings.
- 1.2 - Alternatively a user will be able to add themselves as a **New User** to the system, allowing them to enter their individual mail server settings and account information. In this way the system should be able to deal with multiple users.
- 1.3 - When a user has successfully logged into Me-Mail they will be able to choose to **Send** a new message or **Retrieve** mail.

The functional requirements involved when a user is **sending mail** are as follows:

- 1.3.1 - **Compose** a new message.
- 1.3.2 - **Send** a message to another e-mail address securely.
- 1.3.3 - It could also be possible at this stage to select a particular e-mail address from a short list of the **10 most used addresses**, although this is not implemented into Me-mail for this project.

- 1.3.4 - **Feedback** will be provided to inform the user as to whether a message had been sent successfully or not.

The functional requirements involved when the user is **retrieving mail** are as follows:

- 1.3.5 - User is able to see a **list** of **all** the messages in their mail **inbox**, each represented by the user who sent the message, the subject and the date sent.
 - 1.3.6 – Alternatively the user can choose to view a **list** of the **new** messages in their mail **inbox** only, each represented by the user who sent the message, the subject and the date sent.
 - 1.3.7 - With the list of inbox messages displayed, the user is able to **Delete, Forward** or **Reply** to any particular message. These functions will also be available to the user when one entire message's content is displayed on screen.
 - 1.3.8 - Also at this point the user will be able to choose to **View the Contents** of any message.
 - 1.3.9 – The entire contents of any message should be displayed including any **attachments**.
 - 1.3.10 - **Feedback** may be provided to let the user know how big the message is and how much free memory is available on the device.
-
- 1.4 - The user can Configure their **Account** information by selecting this from the main options menu, the user should be presented with a screen that allows them to edit any of their account settings.
 - 1.5 - The user is able to **log out** of the system at any time by selecting this from the main options menu, hence closing the connection to their e-mail account.

3.3 Non-Functional Requirements

Non-functional requirements describe constraints placed on the system and on the designers which may impact later phases such as the design e.g. cost, time scales, database performance, specific data representations, memory requirements, standards, and so on [Sommerville, 1995]. These requirements are particularly susceptible to change, as they are closely tied to hardware, an area in which advances are so rapid. Again, it is important that non-functional requirements are worded in a way so that they can be verified quantitatively.

Software developers tend to stress the importance of the functional requirements of an application more than the non-functional requirements during development. However this may prove to be an inaccurate decision as in the case of this project. It could lead to a perfectly designed e-mail client implementing all the necessary protocols correctly while lacking the ability to provide users access to their inbox and the retrieval of message content in an acceptable time, without leaving the user waiting long periods of time.

The following have been identified as the key non-functional requirements that are necessary to ensure that the application is both technically and fundamentally sound.

- 2.1 - **Implementation language:** As this project is based on the MIDP profile, it is essential that it is developed using the MIDP J2ME toolkit provided by Sun.
- 2.2 - **Performance & Efficiency:** The application must be able to provide fast access to the user's inbox and also retrieve a message's content in an acceptable time (approximately less than 30 seconds).
 - **Response Time:** The application must be capable of responding to user requests with a minimal delay.
- 2.3 – **Portability:** The system must be platform independent, including any device that is described by the MIDP profile.

- 2.4 – **Scalability:** The system should be easily scalable. This means that the number of users is able to increase without degrading overall system performance.
- 2.5 - **Consistent:** The system must be consistent. Consistency normally enhances the users' possibility for transfer of skills from one system to another. [Nielsen, 1993]
- 2.6 - **Feedback:** It must avoid errors and give good feedback if an error occurs.

3.4 Hardware & Software Requirements

3.4.1 Hardware

As the intended system is to be implemented using Java 2 Micro Edition, then the scope of hardware supported is defined by the individual Configuration and profile used when developing the application. In the case of this project the Configuration is CLDC and the Profile is MIDP, so any device that complies with these standards will be capable of running Me-Mail.

In order for this application to be effective the mobile device must be within network coverage and the user's mail server must be responding. These requirements will be tested when a user tries to use functionality within the system which is dependent on these elements and appropriate feedback will be supplied conveying problems encountered.

The network type that this application is best suited for is the 3G network. Some mobile operators and carriers have manufactured Java-enabled devices for the 2G and 2.5G networks but due to a lack of testing in these environments there exists an uncertainty as to whether the data transmission speeds available on these networks would meet the Performance & Efficiency requirements as outlined above.

3.4.2 Software

In order to run the intended e-mail client application the mobile device must have the KVM virtual machine installed on it. This is the virtual machine for applications developed for devices that are defined in the CLDC.

To develop the application the following software was needed:

- JDK 1.3.0 available from JavaSun J2SE, 2001 as the J2ME toolkit requires JDK 1.3.0 or above.
- MIDP API available from JavaSun J2ME, 2001.
- J2ME toolkit, available from JavaSun J2ME, 2001.

3.5 Methodology Overview

The British Computer Society defines a methodology as ‘a recommended collection of philosophies, procedures, rules, tools, techniques, documentation, management and training for developers of information systems’.

[The British Computer Society, 1998]

What follows is a brief description of various development methodologies with the purpose of choosing one for the project development.

- **Waterfall** approach - this is the traditional approach, derived from other engineering processes. Each activity in the development is represented as a separate phase, each of which must be completed and ‘signed-off’ before the next phase can commence. This model contains the following stages: feasibility study, requirements analysis and definition, systems and software design,

implementation and unit testing, integration and system testing, and finally review and maintenance.

- **Evolutionary** development - the activities of specification, development and validation are mixed so that an initial prototype can be rapidly developed to show the customer. This initial prototype is then refined in an iterative process until it is of an acceptable standard for release. This is known as evolutionary prototyping. Alternatively, the initial prototype may be used just to gain a better understanding of the requirements and customer needs, after which it is discarded. This is known as 'Throw away' or 'exploratory' prototyping.
- **Formal Transformation** - this method formally describes the system specification in a mathematical way. It is then transformed using mathematical methods into a program.
- **Reuse** of components - the assumption is made that part of the system already exists. The next step is then to integrate these components, rather than develop the system from scratch.

3.5.1 Chosen Methodology

This 'classic' approach has its merits and indeed is being used extensively in industry, but it has serious shortcomings, mainly due to its inability to cope with change and iterative development. Evolutionary development or prototyping addresses some of the waterfall model's problems, although it is more suitable in certain circumstances and in certain systems e.g. relatively small systems. However, when combined with a more traditional methodology, prototyping looks like a more feasible approach for modern systems development, and indeed this project.

Therefore to reiterate, this project was underpinned by a methodology combining aspects of the traditional waterfall model and evolutionary prototyping. Those aspects used from the waterfall model will include requirements analysis and design, system

specification, implementation and testing. These stages will lead to the initial prototype. After this initial design has been finalized, the process will be an iterative one, improving upon existing features and adding new functionality as needed. Evaluation and testing will be continual throughout the process. These areas are described in further detail below:

- **Feasibility Study** – First of all information pertinent to wireless application development, J2ME configurations & profiles, J2ME network programming, J2ME user interface design, SMTP & POP3 servers, and wireless devices & networks was collected to decide whether the project was realistic in its objectives. Factors involved included technical expertise, time and resources required. This information was used to consider implementation alternatives.
- **Requirements Gathering** – it was decided that the system should perform a number of core tasks, with a few more left over for possible future implementation if time allowed. These core tasks can be found in section 3.2, although at this stage requirements 1.2, 1.3.2, 1.3.3, 1.3.6, and 1.3.9 are not expected to be completely implemented.
- **Code Design and Implementation** – the code modules were designed and written in Java using the Java Developer's Kit and the Kawa IDE, while the code was compiled using the Ktoolbar from the J2ME toolkit. The implementation starting with the base network connection classes and then the SMTP and POP3 protocols were encapsulated into separate classes.
- **User Interface Design and Implementation** – once the underlying code was working satisfactorily, more attention was paid to the user interface design, keeping in mind basic principles lay down by Nielson and Schneiderman. The interface was implemented using the High-level MIDP UI API as explained in 2.2.5.2., which results in MIDlet portability across MIDP compliant devices.
- **Evaluation and Testing** – at each major milestone, the system was evaluated and tested. This ranged from informal evaluation e.g. asking a lecturer or fellow student for their opinion, to formal e.g. timing server transactions against Performance & Efficiency requirements.

- **Iteration** – once the initial prototype was operational, iterative development continued until the deadline, with each iteration being evaluated and tested. At this stage version control was important so that files were not mixed up with the wrong system version. This was achieved by storing each file version on a backup disk with a version number.

Note: Documentation was an on-going process.

3.6 User Interface

HCI is one of a number of terms used ‘to describe the communication between people and computer systems’, [The British Computer Society, 1998, p80]. HCI is an interdisciplinary approach used to develop systems that are intuitive to users of all backgrounds. A measure of its growth in stature is shown by the fact that ‘the ACM Special Interest Group in Computer Human Interaction (SIGCHI) had more than 6000 members in 1997’, [Schneiderman, 1997, p8]. The key concept of HCI is **usability**. It is made up of five components, namely:

- Learnability
- Rememberability
- Efficiency
- Reliability
- User Satisfaction

[Constantine et al. 1999]

HCI experts endeavour to achieve usability in systems by:

- **Understanding** the factors that determine how people make use of computer technology effectively.
- **Developing** techniques and tools to help designers ensure that information systems are suitable for the activities for which people will use them.
- **Achieving** effective, efficient and safe interaction both in terms of individual human-computer interaction and group interaction. [Preece, 1994]

Therefore, when developing a system, it is vitally important to identify potential users. In today's marketplace, it is essential that software developers pay considerable attention to interface design. In a nutshell, this means the user should not have to adapt to the interface; rather the interface should be intuitive and natural for the user to learn and to use. This project is focusing more on the technology behind the interface. Although it is necessary when developing a user interface for an application that will be used on a device with serious screen size constraints; to respect the limited interface resources available.

3.7 Design

A flow chart of the Me-mail framework is shown below giving a quick overview of the system's flow of control. Any item surrounded by a dotted line represents functionality that will not be implemented into the system initially.

The basic flow of control in Me-mail will follow the structure of this flow chart:

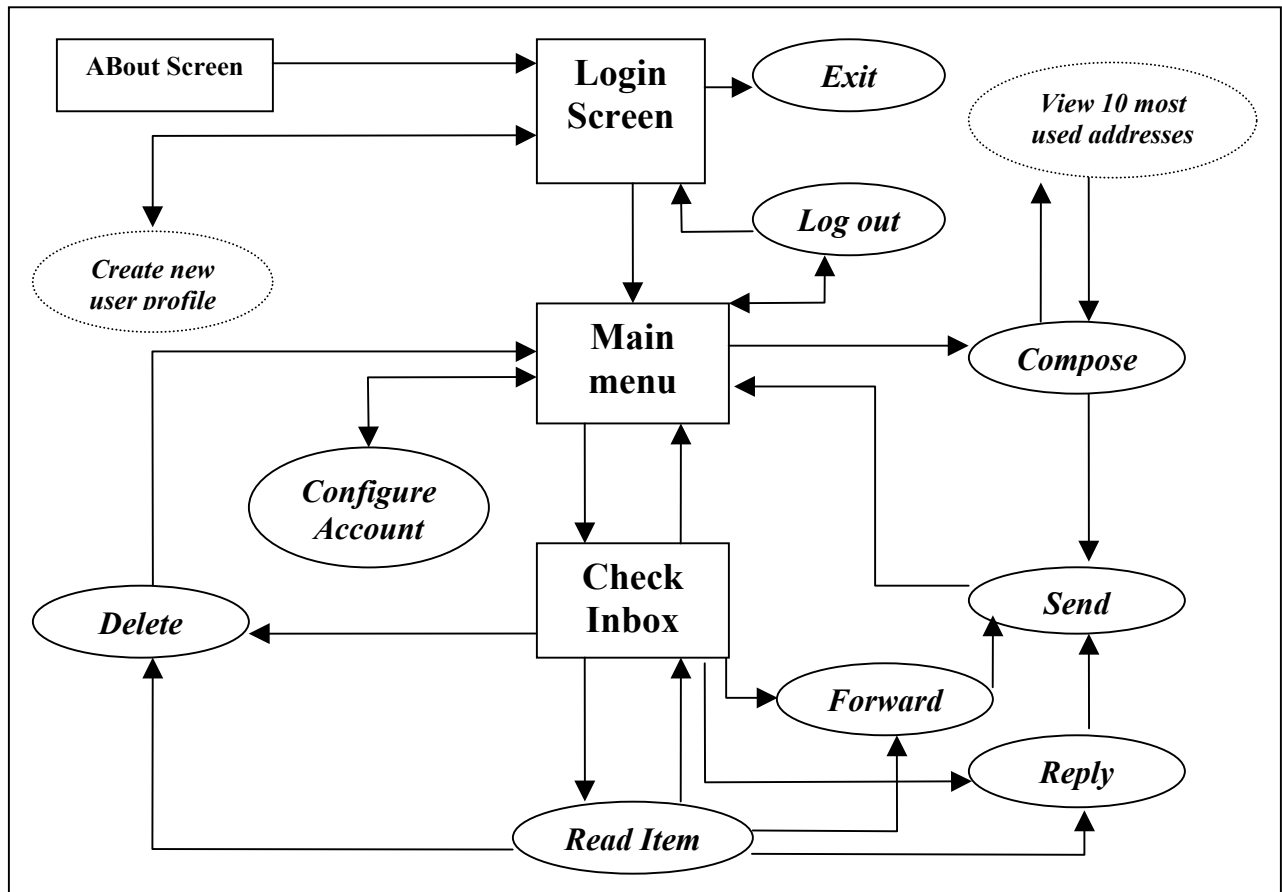
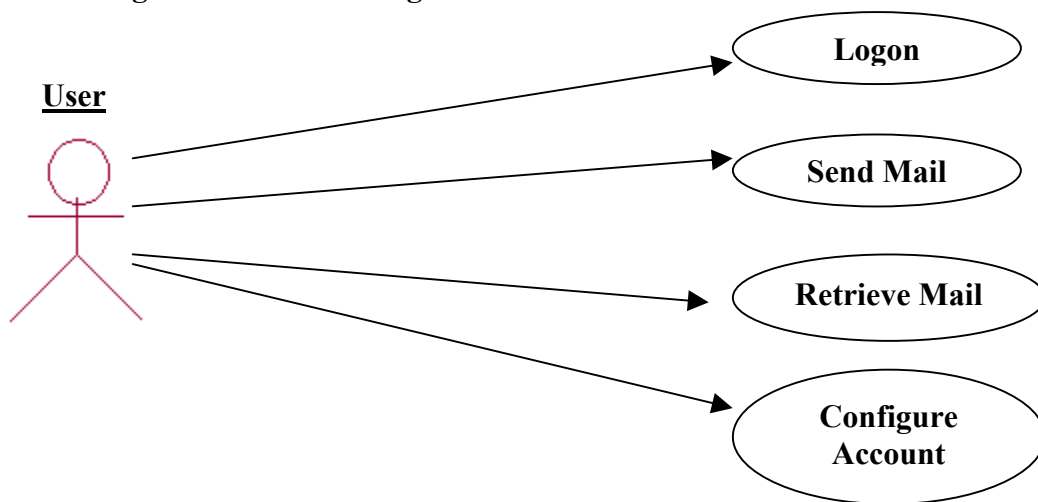


Figure 6: Flow chart for Me-mail.

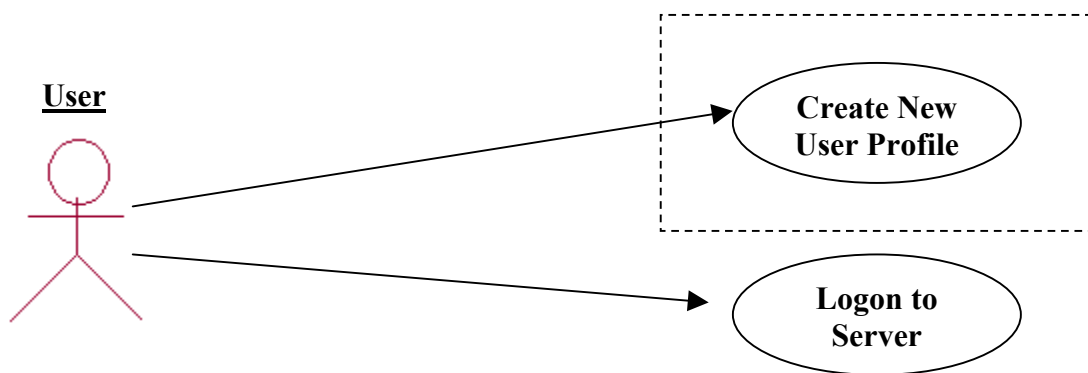
The design of the system is then described using techniques from the Unified Modeling Language (UML), as this design method was encountered previously in an Object Oriented Modeling module and it presents some of the best ways for modeling a system's structure.

Below shows a high-level use case diagram, which represents the typical interactions between actors and the application at a high abstract level. It shows the major functions and tasks that can be performed by a user of the system. Each use case is then broken down into an elementary use case diagram, explaining its functionality in further depth.

- **High-level use case diagram:**



- **Elementary Logon:**



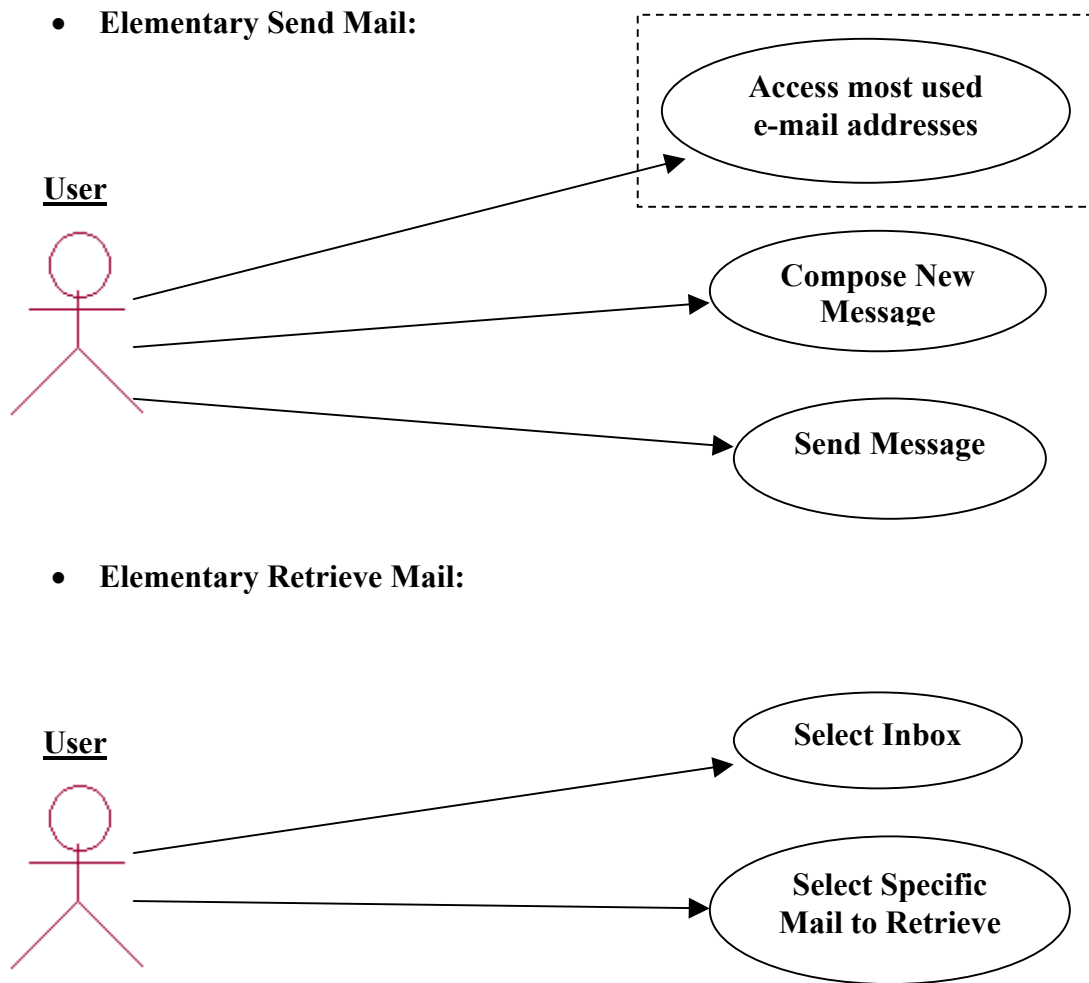


Figure 7: High-level use case diagram & subsequent elementary use case diagrams.

To implement the functionality outlined in the requirements a variety of classes representing various object classes identified within the system are used. The result is an application that allows basic manipulation of an inbox.

Below shows a class diagram, which represents the Object classes that have identified in the system. The attributes and methods implemented in each class are discussed later in the implementation chapter and are included in the appendices.

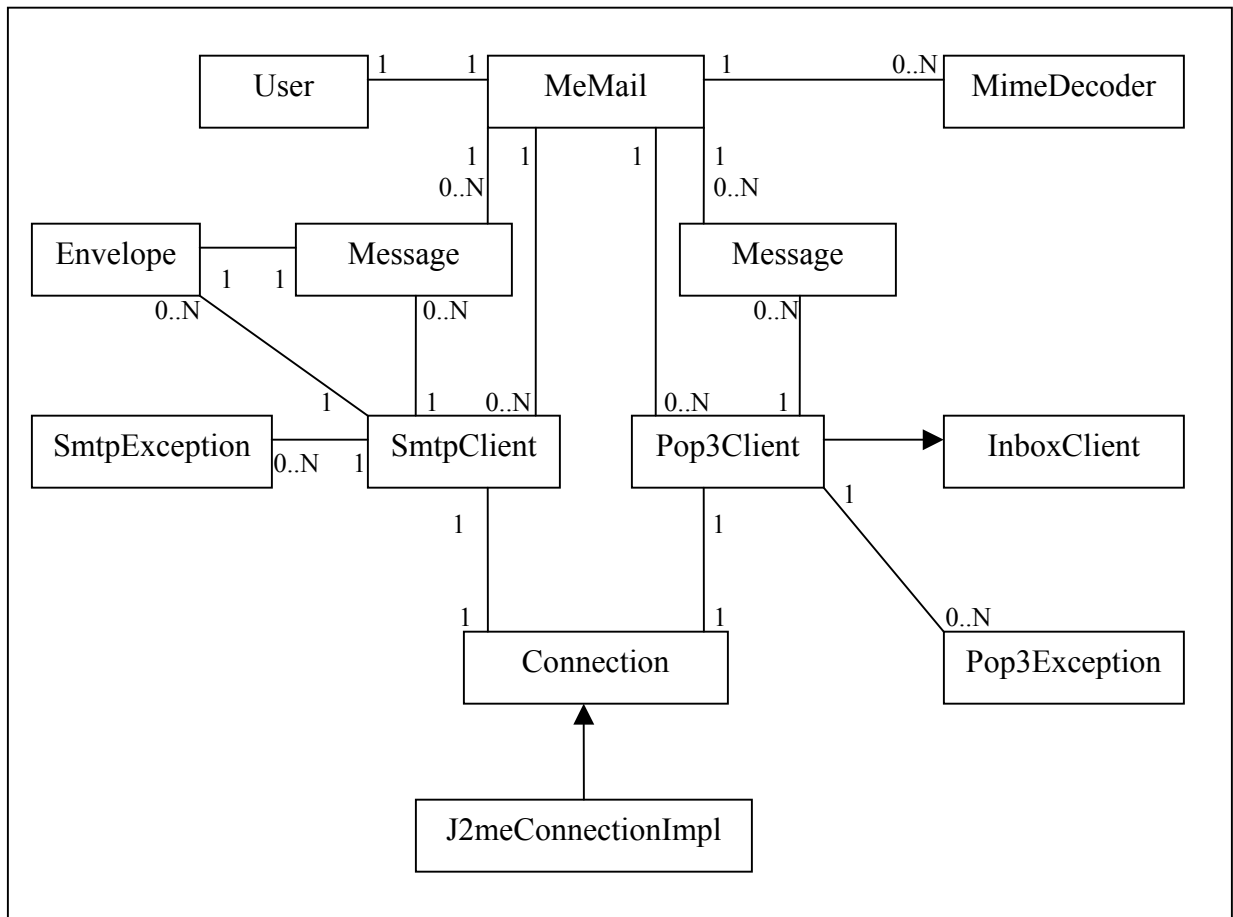


Figure 8: Class diagram of the Me-Mail system.

The main reasons why J2ME was chosen to develop the application are as follows:

- J2ME is an industry standard, backed by a majority of mobile carriers and operators, meaning the application can run on any Java-enabled mobile phones produced by these companies.
- The J2ME solution is better than a similar application developed in native language, as the user can upgrade and install new releases without getting the manufacturer involved, as would be the case if the application was a fixed-feature application developed in the native language of the device.

To send e-mail messages from the application the SMTP protocol will be used. The SmtplibClient class visible in the class diagram above provides the necessary attributes and methods for implementing this.

Due to the fact that the functionality and options that are implemented in the application are reasonably basic POP3 will be employed as the protocol to define the transfer method involved in retrieving messages from the user's mail server, as apposed to IMAP. IMAP does support a much more advanced set of functionality than POP3, but as the aim of this project is to only implement the basic functionality provided by an e-mail client the additional functionality provided by IMAP is not necessary and POP3 will suffice. The Pop3Client class visible in the class diagram above provides the necessary attributes and methods for implementing this.

Despite POP3s inferiority compared to IMAP, the functions supplied by POP3 can be utilised in a way that extra services not defined by the protocol can be implemented with a little extra coding. For example POP3 does not provide any operation to check if there's new mail in the inbox, but this can be done by comparing the previous number of messages in the inbox to the current number, similar to Microsoft's Outlook package and many others that use the POP standard.

POP3 is intended to implement the Offline model for remote mailbox access, but in this project the functions provided by the protocol have been used to design a pseudo-online model for retrieving mail from the server. A precise implementation of the offline model is not feasible due to memory constraints, as it would mean downloading all new messages from the Pop3 server to the PDA/mobile device.

Also if time permits, the scope of the application will be extended to allow for more than one user, enabling anybody to set them-selves up as a user by adding their profile to the application. This would include details of their SMTP and POP3 servers. Initially these settings will be hard-coded into the application for a default user.

Attachments are a very important part of any e-mailing system, but functionality to support this will not be implemented into Me-mail initially. Like the functionality for allowing multiple users, if time allows it will be added to the system later using the MIME standard.

Security is something that an application of this sort must deal with, although it is not anticipated that time will allow for much in-depth research into possible security measures that could be implemented into Me-Mail. Such measures might include digitally signed and encrypted messages. Alternatively some third party software could be used for this purpose.

If a Java-enabled phone is not available for testing the application, or if the data transmission speeds available on current networks prevent the system from being usable on these networks, then the application will be tested using an appropriate emulator running on a standard desktop computer. If the application runs on the emulator, then it's logical to conclude that the same code will also run properly on a Java-enabled wireless device that complies to the CLDC /MIDP standards.

3.8 Summary

This chapter has clearly identified the requirements, both functional and non-functional, necessary to help design a wireless e-mail client application that is suitable from a users perspective and will meet the user's conceptual model of what functionality a system of this nature should provide, while also applying the necessary technological aspects appropriately. Each requirement was individually numbered to ensure traceability through the development. The proposed design followed these requirements closely, which is visible from the diagrams above.

Chapter 4 – Implementation

4.1 Introduction

To recap, the methodology chosen to design the Me-mail application will be a combination of the traditional Waterfall model and Evolutionary development.

Those aspects used from the waterfall model will include the requirements analysis and design, system specification, implementation and testing. These stages will lead to the initial prototype.

This chapter describes in detail the stages involved in the implementation phase of the project. After this initial design has been finalised, the development of the system will be iterated using evolutionary prototyping until a satisfactory prototype has been produced, where all of the user's requirements have been fulfilled to an acceptable level. Each iteration will include implementation, evaluation and testing.

In order to implement the design of Me-mail the functional requirements were reduced to the minimum required to provide a lightweight implementation of an e-mail client. This should ensure the completion of these requirements at least. If extra time is available before the project completion date, this time will be used to implement additional features into the design by iterating the development of the prototype until the updated functional requirements are satisfied.

4.2 Implementation

The Me-Mail MIDlet involved three main distinguishable areas of development:

- **SMTP implementation,**
- **POP3 implementation,**
- **User Interface design.**

These areas will be dealt with in turn in this chapter, explaining the details behind their implementation. Before describing these areas though its necessary to portray how the network connection is dealt with in Me-Mail (An explanation of J2ME networking was given in 2.2.5.3).

In order for an e-mail client to communicate with with a mail server, a socket connection must be established between the two. In J2ME this is achieved using the `StreamConnection` class from the `'javax.microedition.io'` package. The code used for doing this is as follows:

```
StreamConnection socket;  
Socket = (StreamConnection)Connector.open("socket://" + host + ":" + port);
```

(Note: All connections in the Generic Connection Framework are created by one common method, `Connector.open()`).

The above code creates a new socket connection and casts this into the required object class. In this case the `StreamConnection` object is used to establish a socket connection.

In this project an abstract base class for socket connections called **'Connection.java'** is used. The rationale for using this class is the difference in how networking is handled in J2ME and J2SE:

While J2SE has `java.net.socket`, J2ME uses the Generic Connection Framework. Unfortunately both are totally incompatible with each other, so a common abstraction

is established to make Me-Mail work in both environments, that being 'Connection.java'. It provides abstract versions of methods that open and close sockets and return the input and output streams attached to them. It also provides a static factory method that is able to instantiate one of two sub-classes of Connection.java that match the different run-time environments. These sub-classes are called **J2meConnectionImpl** and **J2seConnectionImpl**, and they are loaded "by name" from the j2me or j2se packages, respectively. This is the only way to get rid of compile-time dependencies on these classes.

This method of implementation was chosen to allow for testing of early prototypes in the J2SE environment, preceding the development of User Interface code to support a J2ME device/emulator. The J2SE implementation of the 'Connection.java' class also makes it possible to expand the Me-Mail package at a later date to include a complete J2SE version if desired. At the moment Me-Mail primarily deals with the J2ME Connection class, however, the packages structure is such that it can easily be modified to include the J2SE Connection class (J2seConnectionImpl) and also any UI classes (e.g. AWT/Swing implementations).

An outline of the Connection Class can be seen in **Appendix A** and the outline of the J2me implementation of Connection: J2meConnectioImpl is included in **Appendix B**.

In order to connect to a mail server, the wireless device must initially create a connection with a dial-up Internet Service Provider (ISP). When this connection is established, communication with the SMTP/POP3 mail-server can commence. Figure 8 below illustrates this process.

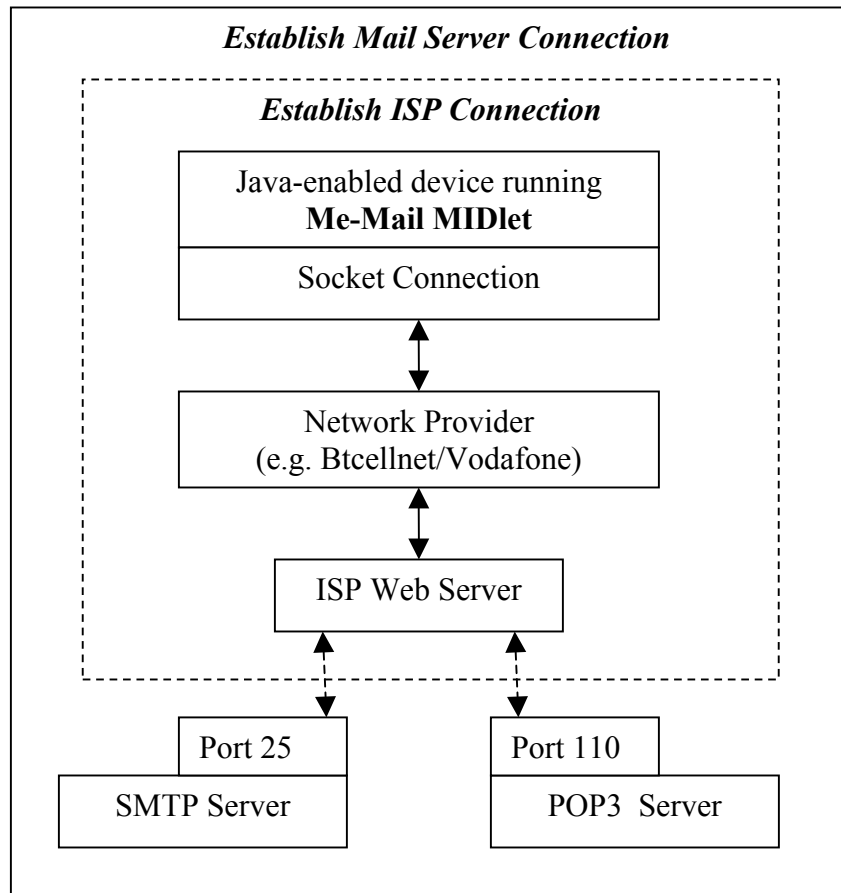


Figure 9: Architecture of network connection.

4.3 SMTP Implementation

The '**SmtplibClient**' class encapsulates the SMTP protocol. This class is used to send email messages as represented by the 'Message' class (designed in accordance with RFC 822) [Appendix D]. After an SMTP session has been established using the open() method, an arbitrary number of messages can be sent using the two variants of the verifySendMessage() method.

- One variant sends a message already contained in an Envelope object [Appendix E], verifySendMessage(Envelope envelope),

- The other creates the Envelope on-the-fly `verifySendMessage(Message message)`. Each SMTP can be closed using the `close()` method.

The method is so called as it returns verification as to whether the message was sent successfully or not in the form of a Boolean value. The skeleton of the `SmtplibClient` class can be seen in **Appendix C**.

The '**Message**' class referenced above embodies an Internet mail message according to RFC 882. From the user's point of view, the class basically consists of two parts:

- An array-like structure holding the message's header lines. Each header line is stored in the textual format that is also used during transmission. To make access to the header data convenient, there are several methods for dealing with header lines as a whole, with names and fields separately and for searching specific header fields.
- An array-like structure holding the message's body lines. Again, several methods are provided to manipulate the body part of the message.

From the classes' internal point of view, both array-like structures are held in a single `Vector` instance to save memory. The skeleton of the `Message` class is contained in **Appendix D**.

The '**Envelope**' class represents an envelope used for sending a message via the SMTP protocol. The envelope object holds information about a message's sender and recipients as used during an SMTP session. It serves more or less the same purpose that an envelope is used for when sending regular mail in the post. Most of the time the envelope data will be identical to the "From:" and "To:" fields of the message header. There are cases, though, where one might want to send a message to a recipient different from the one originally specified in the message. A mailing list is an example of an application that could make use of this feature; postings are forwarded to the subscribers without changing the original message. Allowing the application to be extended in the future to include this functionality is the reason for

separating the envelope from the message itself. The skeleton of the Envelope class is included in **Appendix E**.

One other class associated with the SMTP implementation is **SmtplibException**. This is an exception class for SMTP errors. An instance of this class is raised whenever a protocol error occurs during an SMTP session. This is the case, for example, when one of the message's recipients is syntactically incorrect or when the message doesn't contain any recipients at all, although some of these problems are dealt with by the user interface event handling code in the MeMail class. The receive() method in the SmtplibClient class returns the server response that signalled the error, which should always be a line starting with a 400+ or a 500+ number. The lines starting with a '4' denote a transient error, which gives the client a chance to try the same action again, while a line starting with '5' means a permanent error. The SmtplibException code is contained in **Appendix F**.

4.4 POP3 Implementation

'**Pop3Client.java**' encapsulates the Pop3 protocol as specified in RFC 1939. The class provides a simple interface to a POP3 mailbox. After a session has been established using the open() method, the number of available messages can be queried by calling the getMessageCount() method, and arbitrary messages or their headers can be retrieved from the mailbox using getMessage() or getHeaders(), respectively. Deleting messages is possible using removeMessage(). Each POP3 session can be terminated by a call to the close() method. An outline of this class can be found in **Appendix G**.

Pop3Client implements an interface defined by **InboxClient**. This interface is a general definition of what a client for the inbox of a message store should provide. It serves as an abstraction for an inbox client for the Pop3 or IMAP protocols.

Although, Me-Mail deals with Pop3 in particular as it satisfies the requirements of a thin email client. The contents of this interface are shown in **Appendix H**.

Pop3Exception is the exception class for Pop3 errors in the same way that **SmtplibException** is for SMTP errors. An instance of this class is raised whenever a protocol error occurs during a POP3 session. This is the case, for example, when the given password is incorrect or when the client tries to retrieve a message that doesn't exist. The `receive()` method returns the server response that signalled the error, which should always be a line starting with a '-' sign, according to RFC 1939. The code for `Pop3Exception.java` is shown in **Appendix I**.

As mentioned previously, the `Message` class represents an Internet mail message according to RFC 822. When accessing a Pop3 server, a presumption is made that the server adheres to this RFC standard. In order to download a message from the Pop3 server the `getMessage(int x)` method in the `Pop3Client` class is invoked. This sends the request 'RETR x' to the server. This server replies by sending the message at index 'x' back to the client line by line. After the `getMessage(int x)` method is called, the `receiveMessage()` method is called which creates a new `Message` object and saves the message being downloaded from the server to this `Message` object. When accessing the inbox, the client can also request to only receive the header of a particular message, giving the user enough information to decide whether they wish to download the full message or not.

When an entire message is downloaded from the server and stored as a `Message` object, its content may consist of a variety of constituting MIME (Multi purpose Internet Mail Extensions) parts. A message's type is deduced by obtaining the value from the 'Content-Type:' value in the header field of the message. Frequent MIME parts that work well in the J2ME environment are:

- "text/plain", which means plain ASCII text,
- "image/png" which means that the part holds a PNG (Portable Network Graphics) image.

Both of these types are dealt with by Me-Mail. To aid the process of detecting MIME parts, a class called MimeDecoder is used.

'MimeDecoder.java' is a helper class for decoding MIME data contained in email messages. As the name implies, MimeDecoder allows a kind of read-only view of the MIME parts contained in a message, giving access to textual as well as binary content. Every MimeDecoder works on and thus represents exactly one MIME part, which according to RFC 2045 - RFC 2049 may be either of the following two:

- A single-part, meaning that it's a leaf in the tree-like hierarchy of MIME parts. In this case content may be retrieved directly using the `getBodyLine()` or `getBodyBytes()` methods.
- A multi-part, meaning that it's an inner node in the hierarchy. In that case subordinate parts should be investigated. These parts may again be single-part as well as multi-part.

To start working with a MimeDecoder, a new instance has to be created for a given message, which results in a top-level MIME view of the message. Subordinate parts can then be accessed using the `getPart()` method that serves as a kind of factory method for MimeDecoders representing the lower MIME levels of the message.

The MimeDecoder is an optional component that can be used on a Message if the application demands it. The Message class, on the other hand, is not dependent on the MimeDecoder at all, which means that the Message class can be deployed without having to deploy the MimeDecoder class, too. Despite their independence from each another Me-Mail always uses the MimeDecoder to examine the content of emails.

The internal structure of the class is as follows:

MimeDecoder holds a direct reference to the message's Vector of lines (both header and body) as well as two integer fields telling where the part represented by the MimeDecoder starts and ends. In case of multi-parts, an additional Vector holds all the occurrences of the part boundary string, which makes it quite easy to create additional MimeDecoders for the subordinate MIME

When a new Message object is created using a message obtained from the Pop3 server, the following call creates a new MimeDecoder object for that message with the aim of displaying the message's contents on the device's screen.

- addPartToScreen(new MimeDecoder(onScreenMessage), messageScreen);

An outline of the MimeDecoder class is included in **Appendix J**.

The code for the method invoked above is as follows:

```
private void addPartToScreen(MimeDecoder mime, Form form) {
    if (mime.getPartCount() == 0) {
        // (PNG IMAGE)
        if (mime.getType().equals("image/png")) {
            byte[] bytes = mime.getBodyBytes();
            form.append(Image.createImage(bytes, 0, bytes.length));
        }
        // (x-PNG IMAGE)
        else if (mime.getType().equals("image/x-png")) {
            byte[] bytes = mime.getBodyBytes();
            form.append(Image.createImage(bytes, 0, bytes.length));
        }
        // (PLAIN TEXT)
        else if ((mime.getType() == null) || (mime.getType().equals("text/plain"))) {
            String s = "";
            for (int i = 0; i < mime.getBodyLineCount(); i++) {
                s = s + "\n" + mime.getBodyLine(i);
            }
            form.append(s);
        }
        else {
            form.append("\nUnable to display \"" + mime.getType() + "\" part named " +
                mime.getName() + ".");
        }
    }
    // (MULTI-PART)
    else {
        for (int p = 0; p < mime.getPartCount(); p++) {
            addPartToScreen(mime.getPart(p), form);
        }
    }
}
//FOR
//ELSE
//METHOD
```

4.5 User Interface Design

'MeMail.java' extends the javax.microedition.midlet.MIDlet class from the MIDP package and therefore is the MIDlet equivalent of a main class. The MeMail class provides the required implementations of startApp(), pauseApp(), and destroyApp() as described in 2.2.6. Also MeMail includes the essential public no-argument constructor MeMail() which is the method instantiated by the Application Management Software (AMS) on the MIDP device when the Me-Mail MIDlet is launched.

The MeMail class also implements the CommandListener interface which provides the event processing method:

```
- public void commandAction(Command c, Displayable d){...}
```

Where the Displayable 'd' is the event source. Displayable can be one of the following:

- List, TextBox, Alert, Form (all from the high-level UI API),
- Canvas (from the low-level UI API).

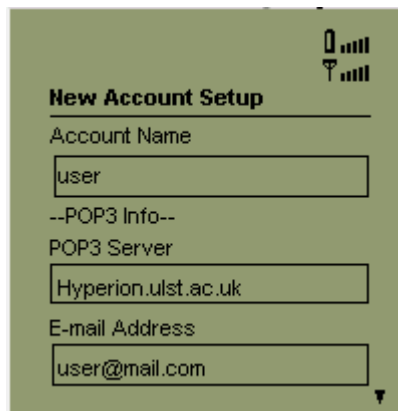
The Command 'c' encapsulates the semantic information of an action/effect. When a Command is activated on any of the Displayable objects, an event is generated and its details are passed to the CommandAction() method, where event-handling code is described.

All the Displayable objects used in Me-Mail are instances of the classes from the high-level UI API, as described in 2.5.5.2. This means that Me-Mail is portable across any MIDP compliant device, as the high-level UI API classes are supported by all these devices. If the Canvas class had been used the application would be useable only on the originally targeted device. Another advantage is that the application could be run on a variety of MIDP devices during the testing phase of the project. The MeMail class is outlined in **Appendix K**.

'User.java' is the object class which stores the attributes that Me-Mail needs to know about a user of the system. When the MIDlet is launched the following screen is displayed:



If the user selects the 'New User' list item, a screen will be displayed which includes text boxes for the various required user details (email address, POP3 server, SMTP server etc.) as shown below:

A screenshot of a mobile device screen titled 'New Account Setup'. It contains three text input fields. The first field is labeled 'Account Name' and contains the text 'user'. The second field is labeled 'POP3 Server' and contains the text 'Hyperion.ulst.ac.uk'. The third field is labeled 'E-mail Address' and contains the text 'user@mail.com'. The status bar at the top shows signal strength and battery level.

When all these details are filled satisfactorily a new user object will be created and this object will be saved to persistent storage (not implemented in final prototype). But, at present the implementation only ever deals with one hard-coded User object and does not utilise persistent storage. Hence, when a new user is created, this new object is instantiated to the already existing User instance. The hard coded user details are included in the User() method which is the only constructor in the class, so every object that is created will include settings for a default email account. This setup was sufficient for testing purposes. Implementation of persistent storage is planned for future versions of Me-Mail, but unfortunately due to time constraints it was not possible to include it yet. A skeleton of the User class can be viewed in **Appendix L**.

4.6 Iterative Development

As earlier stated, after an initial design has been finalised, the development of the system will be iterated using evolutionary prototyping. Iterative development is the final stage in the software's lifecycle. This is when further enhancements are made to the system based on user comments and reaction to the initial prototype. The prototype for this system was exposed to various individuals, including lecturers and fellow students. All those consulted provided excellent feedback and constructive criticism, all of which was fed back into the development. This feedback on the prototype formed an excellent base from which the development could continue.

As there is no final version with evolutionary prototyping, there may be many iterations, with each corresponding to a new release or version (Each iteration includes implementation, evaluation and testing). Hence, it is vital to keep track of these versions to avoid confusion. This was achieved by maintaining a directory structure of all project files, with each version stored in a directory according to when it was created. In this fashion, it was easy to rollback to the previous version as a reference point, if development with the new version was not going according to plan.

4.7 Implementation summary

This chapter dealt with the implementation of the system, initially explaining the hybrid waterfall-prototyping approach followed. The implementation of the various phases involved in developing the system were detailed: SMTP implementation, POP3 implementation, and User Interface Design. This included the process of describing the classes that were used and how they were developed.

Finally the evolutionary development phase was documented. This phase was driven entirely by reaction to the initial prototype, and as new ideas came to light, one possible way of handling changes to a prototype was explained.

Chapter 5 - Evaluation and Testing

5.1 Introduction

This chapter documents the results of testing and evaluating the system. Each requirement specified in section 3.2 was tested, with the user interface evaluation based loosely on Schneiderman's 'Eight Golden Rules of Interface Design' [Schneiderman, 1997].

5.2 Evaluation

There are two forms of evaluation: **Formative** and **Summative**.

Formative has been defined as 'a method of judging the worth of a program while program activities are forming or happening. The focus is on the **process**.' [Bhola, 1990, p16] The purpose of formative evaluation is to provide feedback so as to improve the overall design and development process.

Summative has been defined as 'a method for judging the overall worth of a program at the end of the program activities. The focus is on the **outcome**.' [Bhola, 1990, p16]. This means that testing adds to the sum of the knowledge about the system, rather than alter its design in any way.

Formative evaluation occurred throughout the entire development process. For example, when any new functionality was added such as the ability to deal with attachments, opinions of fellow students and lecturers were sought. Their comments and suggestions were then fed back into the overall process. Summative evaluation occurred at the end of development, with each requirement in turn evaluated. (Although with evolutionary prototyping there is no 'end' as such, for practical reasons development was halted to allow summative evaluation to occur.)

When evaluating a system, both functional and non-functional requirements must be tested to ensure a quality product is produced. The functional requirements are very

specific to the system, but it is vital that they are clear and succinct so that when testing is performed, it is crystal clear whether the test has succeeded or not. It is also vitally important that each test is traceable back to its requirement to ensure that each requirement has been tested.

As for non-functional requirements, this refers to constraints placed on the system and the developers, such as performance, or certain programming languages to be used. The other main issue in systems development is usability. As Scheiderman's rules for interface design were used as a base for testing, it is worthwhile briefly discussing each point:

- **Consistency** – consistent actions should be required in similar circumstances while the same terminology should be used throughout a system. Colour, font, layout, etc should also be consistent in the system.
- **Shortcuts** – frequent, knowledgeable users should not be hindered, therefore hidden keys, macros, keyboard shortcuts etc should be provided.
- **Informative feedback** – for every user action, there should be system feedback, depending on the action's frequency and importance.
- **Closure** – sequences of actions should have a start, middle and end so users are aware when a group of related actions have been completed.
- **Error prevention and handling** – systems should be designed so that users are unable to make catastrophic errors. If errors are detected, the system should offer simple advice and instruction to aid recovery.
- **Reversal of actions** – actions should be reversible. This reduces user anxiety and encourages users to explore the system.
- **Support internal locus of control** – users should be initiators of actions, rather than the responders to actions.
- **Reduce short-term memory load** – this should be reduced to aid ease of learning by adhering to the 'seven plus or minus two' principle [Millar, 1957].

Note: These underlying principles must be tailored to suit the needs of the individual system. Interface issues were not central to the success of this project, so user

interface testing had a lower priority than testing for functional and non-functional requirements.

5.2.1 Functional Requirements Evaluation

Requirement Tested	Result	Comment
1.1	Part-Success	<i>Initially the user will be able to Login, where they have already added themselves as a user in the application and configured their user settings.</i> The user is able to do this successfully, but due to time constraints the application presently only saves one user's details, and these are hard coded as the default system user. The system is scalable and this feature can be implemented further in the future.
1.2	Part-Success	<i>Alternatively a user will be able to add themselves as a New User to the system, allowing them to enter their individual mail server settings and account information. In this way the system should be able to deal with multiple users.</i> A user of the system can add themselves as a New User, but as stated previously the system will not store these details persistently, as this functionality has not been implemented yet.
1.3	Success	<i>When a user has successfully logged into Me-Mail they will be able to choose to Send a new message or Retrieve mail.</i>

		Both of these have been implemented successfully. Code was developed that encapsulated the SMTP and POP3 protocols, hence enabling this functionality.
1.3.1	Success	<i>Compose</i> a new message. When a user chooses to send a message from the main menu a new screen appears on the MIDP device that contains three editable textbox which represent the recipient, subject and content of the email message.
1.3.2	Success	<i>Send</i> a message to another e-mail address securely. When the user selects the send button displayed on the screen described above, the application will create a new Message object and try to send this message.
1.3.3	Fail	<i>It could also be possible at this stage to select a particular e-mail address from a short list of the 10 most used addresses.</i> This option has not been implemented as it requires persistent storage, something which has not been dealt with due to time constraints.
1.3.4	Success	<i>Feedback</i> will be provided to inform the user as to whether a message had been sent successfully or not. Feedback is provided as to whether the message was sent successfully or not.
1.3.5	Success	User is able to see <i>list</i> of <i>all</i> the messages in their mail <i>inbox</i>, each represented by the user who sent the message, the subject and the date sent.

		This information is displayed on a screen that appears after the user chooses to 'Check Mail'. The details of each message are contained as single list item.
1.3.6	Fail	<i>Alternatively the user can choose to view list of the new messages in their mail inbox only, each represented by the user who sent the message, the subject and the date sent.</i> This option has not been implemented as it requires persistent storage. The POP3 protocol does not supply a command to query the server for a list of new message, so the only way to develop this is to save the messages that the user has viewed already. This information could potentially be saved in the User object for each registered system user.
1.3.7	Success	<i>With the list of inbox messages displayed, the user is able to Delete, Forward or Reply to any particular message. These same functions will also be available to the user when one entire message's content is displayed on screen.</i> These options are included in a menu in the Inbox screen where the list of message headers (sender, subject, date, etc...) are displayed. Also this functionality is accessible from the message screen, which contains the full contents of a message.
1.3.8	Success	<i>Also at this point the user will be able to choose to View the Contents of any message.</i> Similar to the Delete, Forward and Reply functionality, this option is included in the menu associated with the inbox, screen.

1.3.9	Part-Success	<i>The entire contents of any message should be displayed including any attachments.</i> The application currently deals with attachments of the following types: • Plain Text, • PNG (Portable network Graphics)
1.3.10	Success	<i>Feedback should be provided to let the user know how big the message is and how much free memory is available on the device.</i> This has been implemented and is an option to the user from the Inbox menu. This allows the user to check the size of any message before choosing to download it entirely. The only problem is that the server returns the size of the message in octets. The conversion from octets to bytes is something that has not been considered yet, as this may be device dependent. But from testing it has been discovered that the ratio of byte:octet is roughly 1:1.
1.4	Success	<i>The user can Configure their Account information by selecting this from the main menu, the user should be presented with a screen that allows them to edit any of their account settings.</i> A screen containing textboxes for each user setting is displayed to allow these values to be saved. If any of the changes are invalid, the appropriate error message is displayed.
1.5	Success	<i>The user is able to log out of the system at any time by selecting this from the main options menu, hence closing the connection to their e-mail account.</i>

		This option is included in the main menu. The event handling code for this command closes any SMTP/POP3 connections that exist.
--	--	---

5.2.2 Non-Functional Requirements Evaluation

Requirement Tested	Result	Comment
2.1	Part-Success	Performance & Efficiency: <i>The application must be able to provide fast access to the user's inbox and also retrieve a message's content in an acceptable time (approximately less than 30 seconds).</i> Response Time: <i>The application must be capable of responding to user requests with a minimal delay.</i> A variety of system operations were performed for this test. Stress tests were applied to the 'View Content' function which downloads messages from the POP3 server. These consisted of trying to download large message possibly with attachments. It was concluded that there was inconsistency in the time required due to varying network speeds. To account for fluctuations in network traffic, these tests were also carried out at different times of the day. During periods of low traffic, message downloads were completed within the specified 30 seconds, while during high levels of traffic (e.g. in the afternoon), times exceeded this, sometimes to unacceptable levels (>1 minute).

		The SMTP related operations were also tested, but due to small amount of communication involved compared to POP3, this section of functionality was well within the server response time. In this case the server response verifies that the message was accepted for delivery.
2.2	Success	Implementation language: <i>As this project is based on the MIDP profile, it is essential that it be developed using the MIDP J2ME toolkit provided by Sun.</i> The code for Me-Mail was compiled using the MIDP compiler accessible from the KtoolBar feature included within the J2MEWTK. This generates all the files (MeMail.jar, MeMail.jad, MANIFEST.MF) necessary to run the Me-Mail MIDlet on a MIDP device.
2.3	Success	Portability: <i>The system must be platform independent, including any device that is described by the MIDP profile.</i> This requirement has been met and even exceeded, as Me-Mail was developed using the MIDP high-level UI APIs which means that the application can be re-used on any device that implements the MIDP profile.
2.4	Success	Scalability: <i>The system should be easily scalable. This means that the number of users is able to increase with out degrading overall system performance.</i> Developing the User class which encapsulates the settings for any one user of the system has fulfilled this requirement. At present the instances of the User class are not stored persistently but this could be easily implemented using the 'javax.microedition.rms' package.

2.5	Success	Consistent: <i>The system must be consistent. Consistency normally enhances the users' possibility for transfer of skills from one system to another. [Nielsen, 1993]</i> Me-Mail is a thin email client that implements the core functionality required. It is consistent by supplying a set of operations similar to any other email client and thus creating a situation where skills can be transferred with ease.
2.6	Success	Feedback: <i>It must avoid errors and give good feedback if an error occurs.</i> Compliance with this requirement was achieved by implementing error catching into any methods that incorporated had a possibility of generating errors especially any that dealt with server communication.

5.2.3 User Interface Tests

These tests are based around the principles laid down by Ben Schneiderman [Schneiderman, 1997] and the interface evaluation checklists specified by Susannah Ravder and Graham Johnson [Ravder and Johnson, 1989]. Usability Evaluation, 2002 was also referred to when choosing an evaluation type.

- *This testing involved asking fellow students to evaluate the system by using it.*

Information was obtained about users' likes, dislikes, needs, and understanding of the system by talking to them, observing them using the system in real work, and letting them answer questions verbally. These types of test runs took place with a total of 10 colleagues, and the following conclusions were drawn:

- Firstly, it was difficult to find the appropriate range of users to evaluate the interface (they were mostly 'expert'), and so this was taken into account whilst attempting to

record the results. Results of this type of evaluation are in the form of noting any relevant information during the observation.

- Secondly, the system performed well in the following areas: Consistency, Error Prevention, User Control, but not so well in these areas: Informative feedback, Visual Clarity and Reversal of Actions. Taking these points in turn then:

- **Informative feedback** – Feedback during message download was the biggest issue encountered. While code was developed to display appropriate feedback during this process, there was difficulty in getting the device to display these screens because of device performance issues. In order to overcome this problem the low-level UI API could be used, but the consequence would be an importable screen as low-level API implementations are device dependent.
- **Visual Clarity** – Due to the nature of MIDP devices and the limited size of the display, visual clarity can be hindered. One such example pointed out by a user during observation is when the Inbox screen consisting of message headers is displayed. Another example mentioned is when an email being viewed contains a large amount of plain text. This can lead to an excessive amount of text being displayed on the screen at one time. This not only necessitates a lot of scrolling but also a lot of time. Due to the way the text is wrapped to fit on screen visual clarity can be seriously diminished.
- **Reversal of Actions** – It was pointed out that when executing a request to download a list of messages in the inbox or an entire message, there is no way to stop this action and return to the previous screen. When communication with a server has started and a request has been sent, the server will continue issuing its response to the clients request until completion. Therefore there is no way to implement any interruption functionality to deal with this problem.

Other less positive comments made by the reviewers included the lack of graphics and icons which would make the application more pleasing to the eye. Graphics were

excluded from the application due to limited device memory and primarily due to screen size constraints. The 'reversal of actions' issue is what bothered most users as when there is a lot of messages in the inbox and the downloading the message headers is proving very time consuming, there is no method for interrupting the download process. Also the same problem applies when downloading a large message.

5.3 Testing

Testing is absolutely essential to ensure a quality product. It falls into two categories, Validation (does the system meet the expectation of the customer?), and Verification (does the system work the way it is supposed to?) [Sommerville, 1995].

During the development of this system, testing was an ongoing process. For example, if a syntax error was encountered in a program, the JDK compiler could take you straight to the line the error occurred in. The code could then be fixed, and the program recompiled. Any changes to the classes necessitated compiling the code using the MIDP compiler. Once the MIDlet was operational, the emulator was re-launched and the relevant function re-tested. The KtoolBar (as described in 2.2.5.1) provided an excellent way of viewing any runtime errors as it includes an output pane where relevant runtime information can be viewed. The KtoolBar preferences can be adjusted to print various types of information to the log pane relating to the MIDlet operation, including:

- Tracing Method invocations,
- Exception tracing,
- Tracing Garbage collection
- Tracing class loading

Also the allocated memory heap size for the emulator can be set using this dialog. This gives the opportunity to stress test the system further by reducing the memory available to the emulator. This provided a very good process for estimating the amount of memory required for an average POP3/SMTP session.

In this way then, run-time errors would come to light. When one did arise, an error code would normally present itself e.g. 'java.lang.OutOfMemoryError', and based on this the problem would be tracked down and the recompilation process repeated.

Typical testing of Me-Mail included running through all the various paths in the system and ensuring the system was working as expected. This testing consisted of checking every screen that Me-Mail displays. What follows is an example of one run through the system including screen shots of the screens tested. The tests executed on each screen included verifying that each button functioned correctly. Each time these tests were applied to Me-Mail a different device emulator was used to ensure compatibility with a range of MIDP compliant devices.

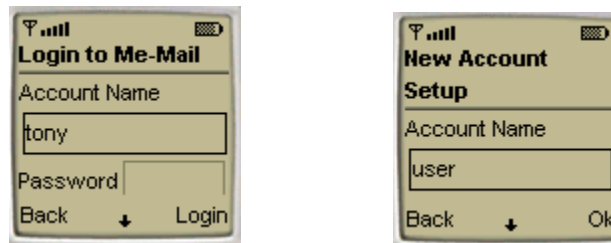
The following screen is the typical screen that is displayed when an emulator is launched. The emulators being used in this test-run will vary for the purposes of increasing the visual clarity of the screen shots. A list of the MIDlets in the loaded MIDlet suite are displayed. In this case Me-Mail is the only MIDlet.



The next screen displayed should be the 'About' screen which gives the details of the application developer. The login menu should be displayed then, which allows the user to: **Login**, Enter their information if they are a **New User**, or **Exit Me-Mail**.



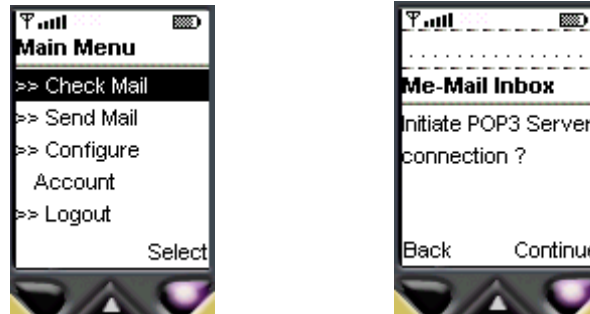
The hard-coded user account is the default account for **login**. The application should not allow a user access to the main menu until they have successfully logged in using a valid user name and password that has been set up using the New User functionality. One can use the **New User** option to create a new user.



If the **Exit** command is selected, the user should be asked to verify that they want to exit. The No button should return the user to the login menu again and the Yes button should close Me-Mail and return the user to the first emulator screen.



When a user hits the Login command in the Login screen the main menu should appear, if the password entered was correct. The main menu should look as follows:



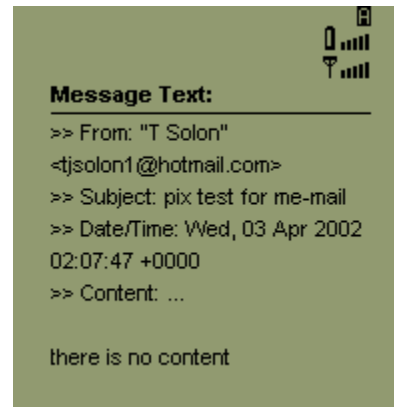
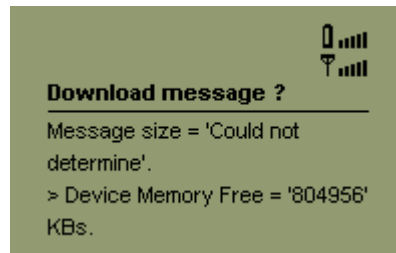
When **Check Mail** is selected, a verification screen should appear asking the user if they are sure they want to initiate the connection to POP3 server.

If continue is chosen a delay ensues while the application communicates with the server requesting access to the inbox and permission to download a list the headers of the messages in the user's inbox. The screen shot below is taken from the BlackBerry emulator as its screen size allows for greater visual clarity.

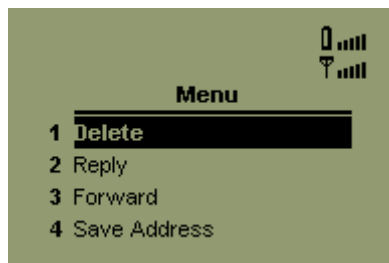


From this screen, one should be able to access the menu displayed along side.

When the **View Content** option is selected the user should be prompted with the size of the message and the available device memory. This is supplied in order to let the user make an informed decision as to whether they wish to download that message or not. If they choose Yes from the menu (hidden in this example) the application should download the entire message.



The options that are available to the user when the entire message is in view are similar to the ones available from the inbox list and are dealt with in the same manner to maintain consistency.



Each of the screens for Reply and Forward are very similar to the screen for the Send message functionality, apart from the screen heading.

During testing, all functionality related to each screen must be executed to ensure they work as expected.

5.4 Summary

This chapter documented the results of the summative evaluation performed and the user interface tests. The results of the evaluation were satisfactory, with each core requirement test rating a success, i.e. each core requirement delivered the

functionality specified. Also, it is felt that the system succeeded in fulfilling the ‘criteria for success’ as described in section 1.4. Although some of the functionality which was considered additional was not completely implemented to specification, these were clearly identified as non-critical for the success of the system. More objective testing by a dedicated tester probably would have highlighted a lot more faults, but under the circumstances, this was the most effective testing available.

The user interface tests consisted of observing each evaluator. These highlighted a few problems, but as specified at the start of this document, this project was inclined more towards the technology, not HCI.

Finally, one complete sample test-run through the Me-Mail application was documented which included screen shots of some of the screens and their associated commands.

Chapter 6 - Conclusion

This project was concerned with the provision of a POP3 based e-mail application that can be used to provide remote mailbox access to users of Java-enabled mobile phones or PDAs. It included research into a wide variety of technologies that either directly apply to wireless device programming, or place it in context. The former includes J2ME and the configurations and profiles it consists of; while the latter included a look at where J2ME fits into the Java family, the origins of current wireless networks and also the operations of electronic-mail systems.

At present, limitations exist that hinder the feasibility of wireless applications. The speed of the currently available wireless networks are the primary concern as they only transmit data at speeds up to 9.6K bps. The solution to this is a faster more reliable network. These requirements are met by the anticipated 3G networks, most of which are expected to in place by 2005. 3G networks will have transmission speeds ranging from 144K bps to 2M bps depending on the movement of the client device. With broadband high-speed digital networks of this nature, there will be a possibility for mobile services that were not feasible previously. Many of these applications will no doubt be new innovative ideas but e-mail is already ubiquitous in today's computing world and its use is predicted to more than double by 2005 [IDC, 2001].

It has already claimed its place in the communication channels for both corporate and personal use and provides us with our primary means of sending and receiving information electronically to and from one another.

With the evolvment of wireless networks and the subsequent possibilities for applications and services on wireless devices it seems inevitable that popular services like e-mail implemented onto these devices will be as popular as their original desktop versions. This project investigated the viability of implementing this service using the currently available network. The conclusion is that a wireless

implementation of this application is indeed feasible, although best suited to faster network speeds that are only available with 2.5G and 3G wireless networks. With the advent of these faster networks many mobile services of this sort will be as viable as their desktop equivalents are today.

6.1 Future Work

When preparing this section, it was quickly realised that it could easily turn into a chapter in its own right. This project attempted to exploit Java enabled mobile devices to make remote mailbox access possible. A future implementation could extend the current application much further. From a design point of view, obviously more emphasis should be placed on usability for later versions.

As stated earlier in 5.2.3, some users commented on the lack of graphics and icons in Me-Mail. To deal with this, small graphics could be included in the application. But these may cause problems on devices with severe screen size constraints. Also more graphical objects could be used on feedback screens, for example when downloading message headers from the server, a gauge could increment one unit each time a message header is retrieved.

More emphasis could have been placed on implementing a security layer into the project. By encoding messages before sending using a reliable algorithm would make this possible; obviously the recipient would require the associated decoder program. Security issues were briefly discussed but were not implemented for this project due to time and resource limitations.

Another feature that could be implemented is a function to determine the parts that comprise an e-mail message as apposed to displaying the entire message on the screen by default. In this way the user could choose to view a particular MIME part. Also more investigation could go into discovering how a larger variety of attachment

types could be handled in the application, especially the more frequent ‘.gif’ and ‘.doc’ files types.

Implementation of multiple users is also an important aspect of any email client. This was omitted from this system due to time constraints. But as discussed in section 3.3, Me-Mail is scalable and this functionality could be easily included in a future prototype. Development of this feature could also include other aspects of persistent storage like: a contact list of email addresses for each user, and also a list of message id’s of previously viewed inbox messages. This would make it possible for a user to select to view new messages instead of their entire inbox folder.

One solution to deal with the visual clarity problems outlined in the UI test results (5.2.3) would be to implement a synthetic talking agent into the email client. This visible character could read email messages to the user or function as a complete interaction agent dealing with every user request [IntelliMedia, 2002]. The problem is that there is no Java APIs available at present that aids access to the audio elements of mobile devices, instead, one would have to develop code in the native language. Such an application would probably be extremely large due to the synthetic agent component. Although, in the future when mobile devices have more memory available the idea will become more feasible. In the long run when OTA (over the air) deployment of wireless applications becomes the norm, an application like this would be accessible to a potentially enormous customer base and with this type of advanced interface would be invaluable to certain users such as the visually impaired.

References

Books:

Avison et al. 1995

Avison, D. E. and Fitzgerald G., *Information Systems Development: Methodologies, Techniques and Tools 2nd Edition*, McGraw-Hill, 1995

Bhola, 1990

Bhola, H. S., *Evaluating literacy for developing projects, programs and campaigns*, UNESCO Inst. for Education, DEE (German Foundation for International Development), Hamburg, Germany, 1990

Constantine et al. 1999

Constantine & Lockwood, *Software for Use: a practical guide to the models and methods of user-centred design*, ACM Press, Addison-Wesley, 1999

Feng et al. 2001

Feng, Y., Zhu, J., *Wireless Java Programming with J2ME*, Sams, 2001

Garg et al. 1999

Garg, V., Wilkes, J., *Principles & Applications of GSM*, Prentice Hall, 1999

Harte et al. 1999

Harte, L., Levine, R., Livingston, G., *GSM Superphones*, McGraw-Hill, 1999

Millar, 1957

Millar, G., *The magical number 7 plus or minus 2: some limits on our capacity for processing information*, Psychological Review, Vol. 63, No. 1, p81-97, 1957

Nielsen, 1993

Nielsen, J., *Usability Engineering*, AP Professional, 1993

Preece, 1994

Preece, J., *Human Computer Interaction*, Addison-Wesley, 1994

Pressman, 1999

Pressman, R., *Software Engineering - A Practitioners Approach*, McGraw Hill, 1999

Schneiderman, 1997

Schneiderman, B., *Designing the User Interface*, Addison-Wesley, 1997

Sommerville, 1995

Sommerville, I., *Software Engineering 5th Edition*, Addison-Wesley, 1995

Sommerville, 1997

Sommerville, I., *Software Engineering*, Addison-Wesley, 1997

The British Computer Society, 1998

The British Computer Society, *A Glossary of Computing Terms 9th Edition*, Longman, 1998

Web sites:

3G Newsroom, 2000

<http://www.3gnewsroom.com>

Amarach, 1999

<http://www.amarach.ie>

BillDay, 2001

<http://www.billday.com/j2me>

EuropeMedia, 2001

<http://www.europemedia.net>

FTP IMAPPOP, 2001

<ftp://ftp.cac.washington.edu/mail/imap.vs.pop>

HowStuffWorks, 2000

<http://www.howstuffworks.com/email5.htm>

IBM, 2001

<http://www6.software.ibm.com/developerworks/education/j-javamail>

IDC, 2001

<http://www.idc.com>

IMAP, 2001

<http://www.imap.org>

IntelliMedia, 2002

<http://www.infm.ulst.ac.uk/~paul>

Intranet Journal, 2001

<http://www.intranetjournal.com>

JavaSun J2SE, 2001

<http://www.java.sun.com/j2se>

JavaSoft, 2001

<http://www.javasoft.com/products/javamail/index.html>

JavaSun J2ME, 2001

<http://www.java.sun.com/products/j2mewtoolkit/>

JavaWorld, 2001

<http://www.javaworld.com/javaworld>

JavaMobiles, 2001

<http://www.javamobiles.com>

NetworkMagazine, 2001

<http://www.networkmagazine.com>

RFC, 2001

<http://www.rfc-editor.org>

Sans, 2001

<http://www.sans.org>

SoftwareDev, 2001

<http://www.softwaredev.earthweb.com>

Sun Developer, 2001

<http://www.developer.java.sun.com>

TeleDotCom, 2001

<http://www.teledotcom.com>

Usability Evaluation, 2002

<http://jthom.best.vwh.net/usability/usable.htm>

WebFoot, 2000

<http://www.webfoot.com>

Appendix A – Connection.java

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public abstract class Connection {

    public abstract void open(String host, int port) throws IOException;
    public abstract void close() throws IOException;
    public abstract InputStream getInputStream() throws IOException;
    public abstract OutputStream getOutputStream() throws IOException;

    public static Connection createConnection() throws Exception {
        String classname;
        if (System.getProperty("microedition.configuration") != null)
            classname = "J2meConnectionImpl";
        else
            classname = "J2seConnectionImpl";
        System.out.println("INFO: Using \"" + classname + "\" for connections");
        return (Connection)(Class.forName(classname).newInstance());
    }
}
```

Appendix B – J2meConnectionImpl.java

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;

public class J2meConnectionImpl extends Connection {
    private StreamConnection socket;

    public void open(String host, int port) throws IOException {
        socket = (StreamConnection)Connector.open("socket://" + host + ":" + port);
    }
}
```

```

public void close() throws IOException {
    if (socket != null) {
        socket.close();
        socket = null;
    }
}

public InputStream getInputStream() throws IOException {
    if (socket != null) {
        return socket.openInputStream();
    }
    else {
        return null;
    }
}

public OutputStream getOutputStream() throws IOException {
    if (socket != null) {
        return socket.openOutputStream();
    }
    else {
        return null;
    }
}
}

```

Appendix C – SmtplibClient.java

```

import java.io.InputStream;
import java.io.OutputStream;
import java.io.IOException;
import java.util.Vector;

public class SmtplibClient {

    private String host;
    private Connection socket;
    private InputStream input;
    private OutputStream output;
    private boolean debug = false;
    private String localhost;

    public SmtplibClient(String localhost) {...}
}

```

```

public void open(String host) throws Exception, IOException,
    ClassNotFoundException, SmtplibException {...}
public void close() throws IOException, SmtplibException {...}
public boolean connected() {...}
private void send(String s) throws IOException, SmtplibException {...}
private String receive() throws IOException, SmtplibException {...}
private String execute(String command) throws IOException, SmtplibException {...}
public void sendMessage(Message message) throws IOException,
    SmtplibException {...}
public boolean verifySendMessage(Message message) throws IOException,
    SmtplibException {...}
public boolean verifySendMessage(Envelope envelope) throws IOException,
    SmtplibException {...}
public boolean getDebug() {...}
public void setDebug(boolean value) {...}
}

```

Appendix D – Message.java

```

import java.util.Calendar;
import java.util.Random;
import java.util.TimeZone;
import java.util.Vector;
import java.io.IOException;

public class Message {
    private Vector lines = new Vector();
    private int headerSize = 0;

    public Message() {...}
    public Message(String from, String to, String subject) {...}
    public String getHeaderLine(int index) throws ArrayIndexOutOfBoundsException
        {...}
    public String getHeaderName(int index) throws ArrayIndexOutOfBoundsException
        {...}
    public String getHeaderValue(int index) throws ArrayIndexOutOfBoundsException
        {...}
    public int getHeaderIndex(String name, int startIndex) {...}
    public int getHeaderIndex(String name) {...}
    public String getHeaderValue(String name) {...}
    public String getHeaderValue(String name, String def) {...}
    public String[] getAllHeaderValues(String name) {...}
    public void setHeaderLine(int index, String line) {...}
    public int getHeaderLineCount() {...}
}

```

```

public int addHeaderLine(String line) {...}
public void insertHeaderLine(int index, String line) {...}
public void removeHeaderLine(int index) {...}
public void setHeaderValue(String name, String value) {...}
public String getBodyLine(int index) {...}
public void setBodyLine(int index, String line) {...}
public int getBodyLineCount() {...}
public int addBodyLine(String line) {...}
public void insertBodyLine(int index, String line) {...}
public void removeBodyLine(int index) {...}
public static String getMachineAddress(String address) {...}
public static String getDisplayAddress(String address) {...}
public static String getCanonicalAddress(String address) {...}
private static String intToStr(int value, int length) {...}
public static String getCanonicalDate(Calendar calendar, TimeZone timezone) {...}
public static String[] getStringElements(String s) {...}
public static String getStringName(String s) {...}
public static String getStringValue(String s) {...}
public static String getRandomString() {...}
Vector getLines() {...}
}

```

Appendix E – Envelope.java

```

import java.util.Vector;

public class Envelope {
    private Message message;
    private String sender;
    private Vector recipients;

    public Envelope(Message message, boolean autofill) {...}
    public Message getMessage() {...}
    public void setSender(String address) {...}
    public String getSender() {...}
    public int addRecipient(String address) {...}
    public void setRecipient(int index, String address) throws
        ArrayIndexOutOfBoundsException {...}
    public int getRecipientCount() {...}
    public String getRecipient(int index) throws ArrayIndexOutOfBoundsException
        {...}
    public void removeRecipient(int index) throws ArrayIndexOutOfBoundsException
        {...}
}

```

Appendix F – SmtplibException.java

```
public class SmtplibException extends Exception {  
    public SmtplibException(String message) {  
        super(message);  
    }  
}
```

Appendix G – Pop3Client.java

```
import java.io.InputStream;  
import java.io.OutputStream;  
import java.io.IOException;  
  
public class Pop3Client implements InboxClient {  
    private Connection socket;  
    private InputStream input;  
    private OutputStream output;  
    private boolean debug = false;  
  
    public Pop3Client() {...}  
    public void open(String host, String user, String pass) throws Exception,  
        IOException, Pop3Exception {...}  
    public void close() throws IOException, Pop3Exception {...}  
    public boolean connected() {...}  
    private void send(String s) throws IOException, Pop3Exception {...}  
    private String receive() throws IOException, Pop3Exception {...}  
    private String execute(String command) throws IOException, Pop3Exception {...}  
    public int getMessageCount() throws IOException, Pop3Exception {...}  
    private Message receiveMessage() throws IOException, Pop3Exception {...}  
    public Message getMessage(int index) throws IOException, Pop3Exception {...}  
    public String getMessageSize(int index) throws IOException, Pop3Exception {...}  
    public Message getHeaders(int index) throws IOException, Pop3Exception {...}  
    public void removeMessage(int index) throws IOException, Pop3Exception {...}  
    public void setDebug(boolean debug) {...}  
    public boolean getDebug() {...}  
}
```

Appendix H – InboxClient.java

```
import java.io.IOException;
public interface InboxClient {
    public void open(String host, String user, String pass) throws Exception,
        IOException, Pop3Exception;
    public void close() throws IOException, Pop3Exception;
    public boolean connected();
    public int getMessageCount() throws IOException, Pop3Exception;
    public Message getMessage(int index) throws IOException, Pop3Exception;
    public String getMessageSize(int index) throws IOException, Pop3Exception;
    public Message getHeaders(int index) throws IOException, Pop3Exception;
    public void removeMessage(int index) throws IOException, Pop3Exception;
    public void setDebug(boolean debug);
    public boolean getDebug();
}
```

Appendix I – Pop3Exception.java

```
public class Pop3Exception extends Exception {
    public Pop3Exception(String message) {
        super(message);
    }
}
```

Appendix J – MimeDecoder.java

```
import java.io.ByteArrayOutputStream;
import java.util.Vector;

public class MimeDecoder {
    private Vector lines;
    private int begin;
    private int end;
    private String type;
    private String name;
    private String encoding;
    private Vector parts;
```

```

public MimeDecoder(Message message) {...}
private MimeDecoder(MimeDecoder parent, int begin, int end) {...}
private void init(Vector lines, int begin, int end) {...}
public String getLine(int index) {...}
public int getBodyLineCount() {...}
public String getBodyLine(int index) throws ArrayIndexOutOfBoundsException
    {...}
public byte[] getBodyBytes() {...}
public int getPartCount() {...}
public MimeDecoder getPart(int index) {...}
public String getType() {...}
public String getName() {...}
public String getEncoding() {...}
public static String getStringName(String s) {...}
public static String getStringValue(String s) {...}
private static int decode(char c) {...}
private static void decode(String s, ByteArrayOutputStream bos) {...}
}

```

Appendix K– MeMail.java

```

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import java.io.IOException;
import java.lang.InterruptedException;
import java.lang.Exception;
import java.lang.Runtime;

public class MeMail extends MIDlet implements CommandListener{

    public User user;

    private Form aboutMemailScreen;
    private List introMenu;

    private Form loginScreen;
        private TextField accountNameField;
        private TextField passwordField;

    private Form newUserScreen;
        private TextField newAccountNameField;
        private TextField pop3ServerField;

```

```

private TextField emailAddress;
private TextField newPasswordField;
private TextField smtpServerField;
private TextField smtpUserName;

private Form checkServerScreen;
    private Ticker ticker;

private Form exitScreen;
private List mainMenu;
private List viewInboxType;
private Form tempInbox;
private List viewInbox;

private Form serverErrorScreen;
    private Form deleteScreen;
    private Form messageScreen;
        private Form verifyGetMessageScreen;
        private Form deleteMessageViewScreen;
    private Form replyScreen;
    boolean messageScreenReplyCommand = false;
    private Form forwardScreen;
    boolean messageScreenForwardCommand = false;

private Form sendMessageScreen;
    private TextField toField;
    private TextField subjectField;
    private TextField contentField;
    private Form abortSendMessageScreen;

private Form replyMessageScreen;
    private Form abortReplyMessageScreen;

private Form forwardMessageScreen;
    private Form abortForwardMessageScreen;

private Form configureAccountScreen;

private Form logoutScreen;

private Command okCommand = new Command("Ok", Command.OK, 1);
private Command yesCommand = new Command("Yes", Command.OK, 1);
private Command loginCommand = new Command("Login", Command.OK, 1);
private Command selectCommand = new Command("Select", Command.OK, 1);

private Command noCommand = new Command("No", Command.CANCEL, 1);

```

```

private Command cancelCommand = new Command("Cancel",
    Command.CANCEL, 1);
private Command backCommand = new Command("Back", Command.BACK, 1);
private Command exitCommand = new Command("Exit", Command.EXIT, 1);
private Command contentCommand = new Command("View Content",
    Command.ITEM, 2);
private Command mailSizeCommand = new Command("Get Mail Size",
    Command.ITEM, 2);
private Command attachDetailsCommand = new Command("Attachment Details",
    Command.ITEM, 2);
private Command deleteCommand = new Command("Delete", Command.ITEM,
    2);
private Command replyCommand = new Command("Reply", Command.ITEM, 2);
private Command forwardCommand = new Command("Forward",
    Command.ITEM, 2);
private Command saveAddressCommand = new Command("Save Address",
    Command.ITEM, 2);

private Command insertAddressCommand = new Command("Insert Address",
    Command.ITEM, 2);
private Command sendCommand = new Command("Send", Command.ITEM, 1);
private Command continueCommand = new Command("Continue",
    Command.ITEM, 1);

private Display display;
private Image memail;
private Message onScreenMessage;

private int newMsgCount;
private Pop3Client pop3;
private SmtplibClient smtp;
Message fwdMessage;

public MeMail(){...}
public void startApp() throws MIDletStateChangeException {...}
public void pauseApp(){...}
public void destroyApp(boolean unconditional){...}

void initAboutMemail(){...}
void garbageAboutMemail(){...}

void initIntroScreen(){...}
void garbageIntroScreen(){...}

void initLoginScreen(){...}
void garbageLoginScreen(){...}

```

```

void initNewUserScreen(){...}
void garbageNewUserScreen(){...}

void initExitScreen(){...}
void garbageExitScreen(){...}

void initMainMenuScreen(){...}
void garbageMainMenuScreen(){...}

void initCheckMailScreen1(){...}
void garbageCheckMailScreen1(){...}

void initTemporaryInbox(){...}
void garbageTemporaryInbox(){...}

void initInboxScreen(){...}
void initServerErrorScreen(String error){...}
void garbageServerErrorScreen(){...}

void initDeleteMessageScreen(String msg){...}
void garbageDeleteMessageScreen(){...}

void initDeleteMessageViewScreen(String msg){...}
void garbageDeleteMessageViewScreen(){...}

void initMessageScreen(int number){...}
void garbageMessageScreen(){...}

void initVerifyGetMessageScreen(int index){...}

void initSendMessageScreen(String to, String subject, String content){...}
void garbageSendMessageScreen(){...}

void initAbortSendMessageScreen(){...}
void garbageAbortSendMessageScreen(){...}

void initReplyMessageScreen(String to, String subject, String content){...}
void garbageReplyMessageScreen(){...}

void initAbortReplyMessageScreen(){...}
void garbageAbortReplyMessageScreen(){...}

void initForwardMessageScreen(String to, String subject, String content){...}
void garbageForwardMessageScreen(){...}

```

```

void initAbortForwardMessageScreen(){...}
void garbageAbortForwardMessageScreen(){...}

void initConfigureAccScreen(){...}
void garbageConfigureAccScreen(){...}

void initLogoutScreen(){...}
void garbageLogoutScreen(){...}
public void garbageCollection(){...}

//EVENT HANDLING METHOD
public void commandAction(Command c, Displayable d){...}

private void addPartToScreen(MimeDecoder mime, Form form) {...}
public static void showMimeInfo(String prefix, MimeDecoder mime) {...}
} //class

```

Appendix L – User.java

```

public class User {
    String accountName;
    String popServer;
    String emailAddress;
    String password;
    String smtpServer;
    String smtpUser;
    String contactList[];
    String msgIds[];
    int msgCount;

    public User(){
        this.accountName = "user";
        this.popServer = "pop.mail.com";
        this.emailAddress = "user@mail.com";
        this.password = "password";
        this.smtpServer = "smtp.mail.com";
        this.smtpUser = "user@mail.com";
    }
    public User(String accountName){...}
    public User(String accountName, String popServer, String emailAddress,
        String password, String smtpServer, String smtpUser){...}
    public String getAccountName(){...}

```

```
public void setAccountName(String accountName){...}  
public String getEmailAddress(){...}  
public void setEmailAddress(String emailAddress){...}  
public String getPassword(){...}  
public void setPassword(String password){...}  
public String getPopServer(){...}  
public void setPopServer(String popServer){...}  
public String getSmtpServer(){...}  
public void setSmtpServer(String smtpServer){...}  
public String getSmtpUser(){...}  
public void setSmtpUser(String smtpUser) {...}  
public int getMsgCount(){...}  
public void setMsgCount(int msgCount){...}  
}
```