NTNU

Innovation and Creativity

Development of collaborative applications for mobile phones

Implementation of the voice messaging system (VMS) using the Peer2Me framework

Hassan Syed Shah

Master in Information Systems
Submission date: June 2007
Supervisor: Alf Inge Wang, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

This master thesis will investigate the development of an innovative artifact for voice messaging in an adhoc mobile environment and peer-to-peer mobile networks using the Peer2Me framework. Peer2Me is a framework for developing mobile collaborative applications on mobile phones.

Assignment given: 15. January 2007
Supervisor: Alf Inge Wang, IDI

Abstract

This study presents the implementation of a voice messaging system using the Peer2Me framework. Voice messaging system (VMS) is an attempt for a new era of communication which is an intuitive-to-use service that adds an emotional and personal dimension to messaging. It enables the user to send voice messages in a peer-to-peer network.

One of the objectives of designing VMS is to use the Peer2Me framework which is a framework for developing mobile collaborative applications on mobile phones. For this purpose an initial background study of the framework, central concepts, related technology and state of the art was carried out.

We started with the realization of the idea of the VMS by defining its functional and quality requirements, software architecture and high level design. The implementation was carried out in MIDP/J2ME. The application was tested throughout the implementation process and a system test was performed on real phones on completion of the implementation phase.

At the end we evaluated our work, discussed the problems we encountered, answered our research questions, gave our conclusions and described further work that could be carried out on the VMS.

(All source code of the VMS is attached along with this report).

Contents

Part I

Introduction

Chapter 1

Introduction

The pace of development in electronics, telecommunications, computers and broadcasting have been rapid and spectacular during the past decade. This tremendous leap in technological development has increased both the use and role of mobile devices in our daily life. It is changing the way we live and work. Not very long ago, mobile phones were costly devices used exclusively by buisness users, but today they have become an affordable everyday device used by the old, young adults and even children. According to a rough estimate, about 90% of Norwegian school children own a mobile phone in their fifth grade [44]. The internet and mobile phones are now an integral part of our life. In addition to the traditional use of making calls while on the move, mobile phones are increasing being used for checking weather forecast, stock market forecast, news, email, web surfing, chatting and the recent context and location aware applications. The possibilities are increasing with the technology leap in wireless broadband communications. The WiFi net in Mountain View, USA and the Wireless Trondheim Project [44] in Trondheim, Norway are the future hybrid networks of wireless communication. This leap in technology gives rise to new customer services and innovative buisness models for both existing and new buisnesses in the society. In the mean while, the global trend in buisnesses is also changing from a standardized production to highly customized products, tailored to customer needs. This is the case for mobile phones as well, where users are no more dependent on the functionality provided by the manufacturers, but can customize their phones by downloading new applications, developed by third party developers. It won't be long before we can use our mobile phone as a common user interface for most of our day to day and leasure activities such as shooting a presentation right off our mobile phones at work, or using it as a GPS navigator while on a trip.

Today, Java is supported by most modern mobile phones and Java 2 Platform, Micro Edition (J2ME), is proving to be de facto standard. Windows Mobile - based smartphones are also increasing in number. These mobile phones support applications developed on the .NET Compact Framework and some even support J2ME applications. With J2ME and personal area networks like bluetooth, software developers can create useful and sensible applications, promoting informal ad-hoc collaboration [39]. Kraut, Fish, Root and Chalfonte defines informal collaboration as:

“Informal collaboration, opportunistic or spontaneous, lacks predefined agenda and schedule, is short, and involves random participants” [37].

Peer2Me is a framework which lets the developer create applications based on the principle of mobile ad-hoc communication. Framework is defined as:

“A set of classes that embodies an abstract design for solutions to a family of related problems” [38].

Peer2Me abstracts the network layer, making the communication semi-transparent to the developers. It’s aim is to facilitate the development of powerful applications that comunicate wirelessly.

The Peer2Me framework used in this thesis is in its second version, which was redisgned by Kim Petter Saxlund and Tommy Bjørnsgård [40].

My aim is to explore the possibilities of developing a Voice Messaging System (VMS), utilizing the features of the framework. The VMS will be intended for Java enabled mobile phones supporting CLDC 1.1 and MIDP 2.0. Since the VMS will provide functionality for recording audio, it requires phones based on Java Plateform 6 or onwards. The VMS will be a new concept in peer to peer voice messaging over mobile phones.

Chapter 2

Motivation

Short message service (SMS) or text Messaging has been a very popular application in mobile phones. It allows people to communicate in an asynchronous fashion by sending text between mobile phones, other handheld devices and even landline telephones. Short message services are developing very rapidly throughout the world and its popularity can be inferred from the fact that the term “texting” has entered the common lexicon.

The Voice messaging service (VMS) is an attempt for a new era of communication which is an intuitive-to-use service that adds an emotional and personal dimension to messaging. It enables the user to send messages to a destination using voice media. Voice itself can be packaged and sent through the network, so that it reaches its marked “address”.

Communicating by voice message will increase the understandability of the message. By taking speech qualities such as the tone, accent, mood etc. into account, the reciepent of the message will be in a better position to understand the overall context of the received voice message as compared to a text message, thereby increasing workspace awareness and trust in collaboration [33].

The time taken by recording a voice message is considerably shorter than typing in a text message and moreover, the sender of the message is free from language restrictions of the mobile phone keypad. It will also be very useful for disabled users, having difficulty with typing messages.

Chapter 3

Problem Definition

Today there are voice messaging systems for phones or similar but it has not been fully explored in an adhoc mobile environment and peer-to-peer mobile networks. The VMS system will focus on providing an innovative voice messaging solution in such environments. A goal of the Peer2Me project is to assist developers in making new mobile applications [40]. This project will also serve the purpose of evaluating this goal.

Chapter 4

Project Context

The VMS project started as a Master Project by Hassan [42] in the fall semester of 2006. It is continued as a Master Thesis by Hassan [43] in the Spring of 2007.

As a whole, the project is a part of MOWAHS, MObile Work Across Heterogeneous Systems, which is a basic research project performed in cooperation with the Software Engineering and the Database Techonology groups of The Department of Computer and Information Science (IDI) at The Norwegian University of Science and Technology (NTNU).

MOWAHS superior goals are:

1. Helping to understand and to continuously assess and improve work processes in virtual organizations.
2. Providing a flexible, common work environment to execute and share real work processes and their artifacts on a variety of electronic devices (from big servers to small PDAs).
3. Disseminating the results to colleagues, students, companies, and the community at large.

As Lund and Norum stated in [35], the Peer2Me framework upon which the voice messaging system is based, is mainly contributing to the second goal of MOWAHS. This project will continue supporting this goal. The Peer2Me framework was developed to help the developer in developing applications based on the principles of mobile ad-hoc communication. As stated before the framework is in its second version having undergone alot of rework by Kim Petter Saxlund and Tommy Bjørnsgård [40]. The Peer2Me framework will be used in the development of the voice messaging system application. This thesis will also contribute to the third goal, but it will focus more on software engineering than collaboration technology.

Chapter 5

Readers Guide

This chapter gives an overview of the different parts and chapters of this thesis report. The contents of this report vary a lot, both with respect to focus and concepts. Some parts of the report might be more interesting to some readers, than others. In order to increase the readability of the report, we will therefore present a brief outline of each chapter.

If you are not interested in reading the whole report, you should read the part that is most relevant to your interest:

Readers interested in the problem domain: Should read Part I along with Part III. Part III describes essential concepts and technologies along with a chapter, state of the art, presenting other projects related to the problem domain.

Readers interested in Research and Development methods: Should take a look at the Part II. Along with the research methods and research questions, this part also describes the development tools and methods.

Developers interested in the VMS application: Should read and understand Part IV and Part VII. These parts present all essential information about the application, beginning with a description. It also describes the functional and quality requirements along with the software architecture, design, implementation, testing and evaluation of the system and the source code.

5.1 Chapter description

In order to give an idea of the contents of this report to the reader, an outline of the chapters is presented here:

Part I - Introduction

Chapter 1 - Introduction This chapter describes the introduction of the project.

Chapter 2 - Motivation This chapter describes the motivation behind the project.

Chapter 3 - Problem Definition This chapter describes the problem definition.

Chapter 4 - Project Context This chapter describes the context of the project.

Chapter 5 - Readers Guide This chapter presents an outline of the report describing the chapters.

Part II - Research and Development Method & Questions

Chapter 6 - Research Methods This chapter presents the research methods used in this project.

Chapter 7 - Research Questions This chapter describes the research questions that are derived from the problem definition and motivation.

Chapter 8 - Development Method, Tools and Software This chapter presents the development methods, tools and software used to create this report and the VMS application.

Part III - Preface

Chapter 9 - Central Concept This chapter deals with the central concepts related to the problem domain. These concepts should be understood in order to grasp the background and intention behind the application.

Chapter 10 - Technology This chapter presents an update on relevant technology.

Chapter 11 - Peer2Me Background This chapter presents a chronological description of the history of the Peer2Me framework.

Chapter 12 - State of the Art This chapter presents similar projects and solutions as the VMS, and a comparison.

Part IV - Own contribution: Voice Messaging System VMS

Chapter 13 - Description This chapter gives a describtion of the VMS system.

Chapter 14 - Requirements This chapter describes the functional and quality requirements of the VMS.

Chapter 15 - Software Architecture This chapter describes the high level software architecture of the VMS.

Chapter 16 - Design This chapter presents the actual design of the VMS , with detail on its different components.

Chapter 17 - Implementation This chapter describes the implementation process of the VMS and the methodology used.

Chapter 18 - Testing This chapter describes the testing performed on the VMS and its outputs.

Chapter 19 - Results & Evaluation This chapter gives an evaluation of the VMS application. It answers the research questions given in Chapter 7. It also evaluates the functional requirements given in Section 14.1.

Part V - Conclusion and Further work

Chapter 20 - Conclusion This chapter gives a conclusion of the report.

Chapter 21 - Further Work This chapter gives a describtion of further work, that could be done to further enhance the functionality of the VMS.

Part VI - Bibliography

Bibliography The bibliography includes all the references that have been taken in the thesis.

Part VII - Appendix

Appendix The appendix contains the source code of the VMS system.

Part II

Research and Development Method & Questions

Chapter 6

Research Methods

This chapter describes the research and development methods that will be used for the development of the voice messaging system VMS. There are a variety of procedures and techniques that attempt to provide an orderly, systematic way of developing software.

6.1 Research Method

The three main research approaches that are commonly used for doing experiments in software engineering as described by Basili in [27] are:

6.1.1 Engineering Method (Scientific)

The Engineering method is characterized by observing existing solutions, proposing better solutions, building/developing, measuring and analyzing , and repeating the process until no more improvements appear possible. This method is an evolutionary approach were the software being developed can go through experimental phases in order to find the best solution. This method can also involve analysis of older versions of the software so that improvements can be made. This method is typically used to find better methods for structuring large systems. Analysis and measurement is crucial for the success of this method.

6.1.2 Empirical Method (Scientific)

The empirical method is a statistical method and is characterized by proposing a model, developing statististical/qualitative methods, applying it to case studies, measuring and analyzing, validating the model and repeating the procedure. This approach is classified as a revolutionary method which can either propose a model based on an existing one, or suggest an entire new model. This method is typicallly used when comparing a new technology against an old technology. Analysis and measurement is crucial for the success of this method as well.

6.1.3 Mathematical Method (Analytic)

The mathematical method is based on mathematical and formal methods for doing experiments. A formal theory is developed and compared with empirical observations to get results. The mathematical method is usually used to find better formal methods and languages.

6.1.4 Chosen Research Method

The method we have chosen to use in this project is the Engineering method. It will include the use of the existing Peer2Me framework and observation of applications build on it and analyzing applications similar to the VMS. Since this project is being carried out as a Masters thesis, it is very time-consuming and expensive to perform full-scale research experiments. In the chosen research method, we will develop a new and innovative artifact to extend the boundaries of human and organizational capabilities. The idea of the new artifact will be generated by examining the current available technologies in mobile phones and the need of users. It will then be elaborated by defining the requirements and initial design and architecture. A prototype system will be created which will then be tested and evaluated. The motivation would be to explore innovative ways/solutions in peer to peer mobile communications.

6.2 Background Research

Since the application will be developed by utilizing the Peer2Me framework, it is very important to have background knowledge about the framework. This will be done by reading the Peer2Me documentation found in the master thesis by Kim Petter Saxlund and Tommy Bjørnsgård [40]. In addition, a thorough knowledge on J2ME, Bluetooth and mobile applications technologies and similar systems in general will be required.

Chapter 7

Research Questions

The scope of this thesis is from the application developers perspective, who uses the Peer2Me framework in his/her applications. The relevant research questions for the project will be:

RQ1 - How easy and convenient the Peer2Me framework is to utilize for the application developer?

RQ2 - To explore the feasibility of the Peer2Me framework for the development of voice/mp3 messaging/file transfers applications like VMS.

RQ3 - Discover technical limitations in the Peer2Me framework.

RQ4 - Identify possible future extensions to the Peer2Me framework.

RQ5 - How the VMS can be used to support mobile collaboration?

7.1 How to find answers to these questions

During the implementation of the VMS, all problems and issues will be noticed, and if applicable, used to answer the research questions described in Chapter 7

Chapter 8

Development Method, Tools & Software

This chapter describes the development method, tools and softwares that will be used for the development of the VMS.

8.1 Development Method

This section describes the development method that will be used for the development of the VMS.

8.1.1 MVC Architectural Pattern

The MVC architectural pattern factors an application into three different parts: a model, a view, and a controller. The model is the data held by the application and the code that deals specifically with the data. In a text editor, for example, the model is the text being edited. The view is a representation of the model, typically drawn on a screen but not necessarily limited to display output. Although there is only one model, there can be more than one view and each view can represent the model in different ways. A spreadsheet, for example, can display data by using a grid-based tabular format or as a chart. Each is a different view of the same basic data. The controller interprets external input, usually from a user, to modify the model or the view [31]. Figure 8.1 shows the MVC architectural pattern.

8.1.2 Waterfall Model

The waterfall model is often considered the classic approach to systems development life cycle. It is a sequential software development model in which the development is

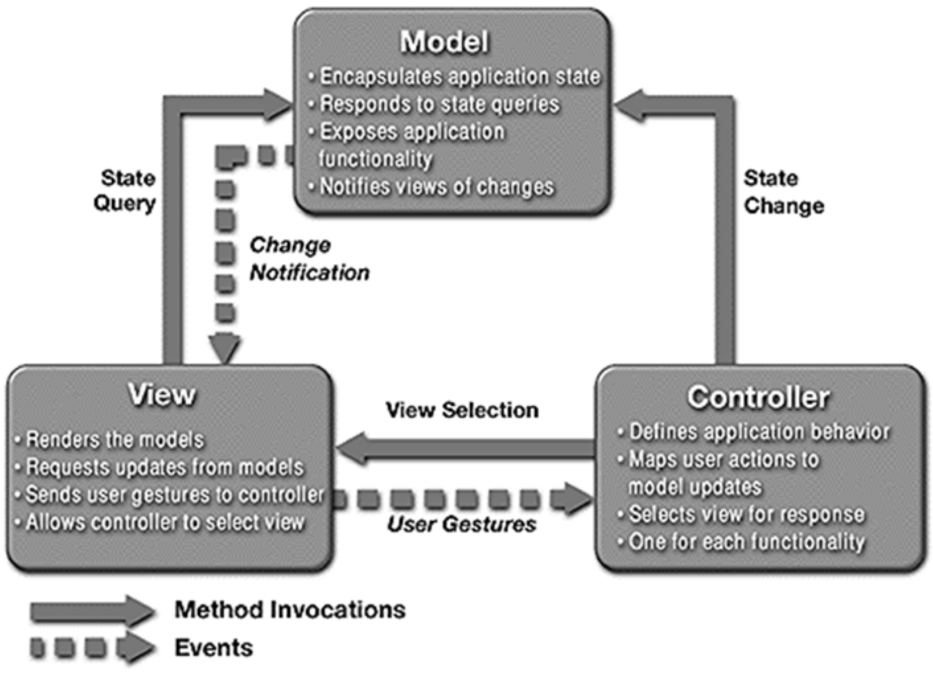

Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
Change Notification
State Change
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

Figure 8.1: Model-View-Controller technique [26].

seen as steadily flowing downwards through the requirements analysis, design, code (implementation), testing (validation), integration and maintenance see Figure 8.2.

Waterfall development has distinct goals for each phase of development:

Requirements analysis - Specification of the problem along with desired objectives (goals) and identification of constraints. This document clearly defines the product function.

Design - Translation of system specifications into software representation. It is concerned with the software architecture, data structure, algorithmic detail and interface representations.

Code (implementation) - Translation of design into the software domain. Detailed documentation from the design phase can significantly reduce the coding effort.

Testing (validation) - Testing focuses on making sure that any errors are identified and that the software meets its required specification.

Integration - Integration of all program units to make the complete system. After this stage the software is delivered to the customer for acceptance testing.

Maintenance - Software updated to meet changing customer needs, changes in external environment, correct previously undetected errors and enhancing the efficiency of the software.

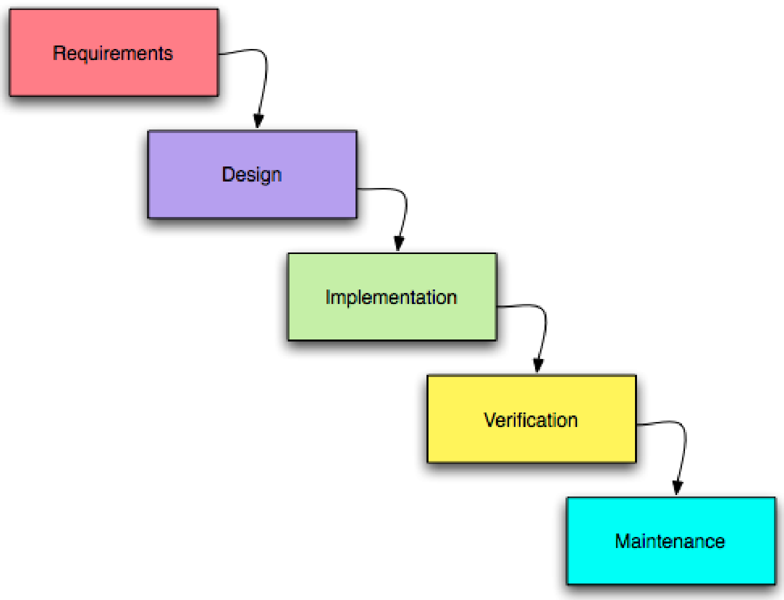


Figure 8.2: The waterfall model.

There are variations in the way the waterfall model is used. It can either be used as a strict step by step process with no turning back between different phases [22] or a lenient one, in which you can move back in feedback loops to allow for corrections to be incorporated in the model. For example a problem/update in the design phase requires a 'revisit' to the requirement specifications phase [21]. When changes are made at any phase, the relevant documentation should be updated to reflect that change. We will follow the lenient model to allow for incremental changes.

The initial depth study done by Hassan [42] focussed on the background work carried out so far regarding the Peer2Me Framework and the related technologies. It also focussed on the requirements analysis and design of the VMS. This master thesis reworks the whole study previously done on the VMS, with more focus on the design, implementation, testing and evaluation of the system. The development tools used are described in Section 8.2.

8.2 Development Tools & Software

This section describes the development tools used in the development of the VMS application and this report.

8.2.1 Eclipse With Plugins

Eclipse [2] is a an open source development platform with lots of available plugins for different purposes, e.g. writing LaTeX documents, creating applications for mobile phones etc.

TeXlipse is a plugin for writing LaTeX documents in Eclipse. It includes features like syntax highlighting, command completion and bibliography completion [12].

EclipseMe [11] is a plugin for developing J2ME MIDlets in Eclipse. A Java Wireless Toolkit must be installed on the system for the plugin to work.

8.2.2 Miktex

MiKTeX is an implementation of TeX and related programs for Windows on x86 systems [7]. The MiKTeX distribution contains many features including the pdfTeX compiler which generates a pdf document. The compiler is used to produce this document.

8.2.3 Emulators

It is cumbersome to always use a mobile phone for testing the applications during development. Therefore, emulators can be used instead. There are several emulators available and each mobile phone producer has its own toolkit.

Sun Wireless Toolkit

The Sun Wireless Toolkit [18] is a standard toolkit that can be used to test applications that are based on J2ME's Connected Limited Device Configuration (CLDC) and Mobile Information Device Profile (MIDP).

Sony Ericsson J2ME SDK

The Sony Ericsson J2ME SDK Toolkit [17] is a standard toolkit provided by Sony Ericsson that can be used to test applications that are based on J2ME's Connected Limited Device Configuration (CLDC) and Mobile Information Device Profile (MIDP) on a series of Sony Ericsson phone emulators.

Part III

Preface

This part will explain the central concepts about peer-to-peer (P2P) systems and technologies. Also, some basic information about the Peer2Me framework is explained. The part ends with a description and comparision of some state-of-the art projects and solutions similar to the VMS.

Chapter 9

Central Concepts

This chapter describes the central concepts used in the project.

9.1 Peer to Peer Networks

P2P networking has gained tremendous interest worldwide among both internet surfers and computer networking professionals. The P2P acronym technically stands for peer to peer. Webopedia [13] defines P2P as ”A type of network in which each workstation has equivalent capabilities and responsibilities. This differs from client/server architectures, in which some computers are dedicated to serving the others.” Peer-to-peer networks are generally simpler but they usually do not offer the same performance under heavy loads. The P2P network itself relies on computing power at the ends of a connection rather than from within the network itself.

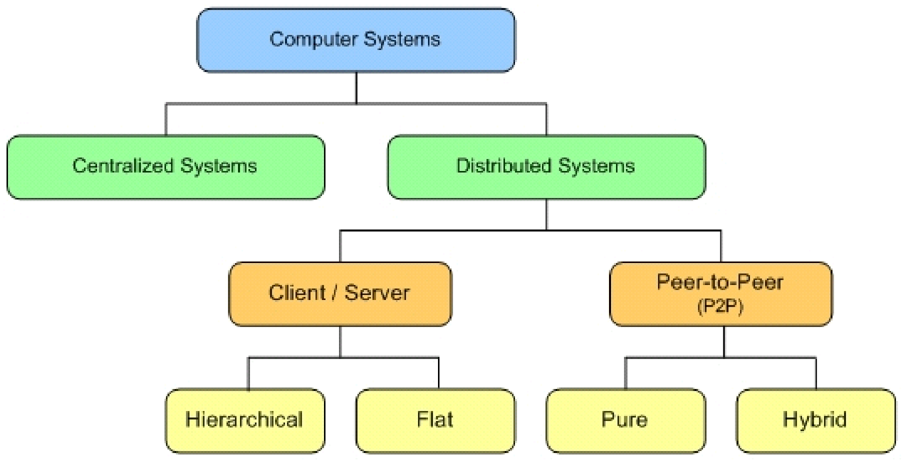


Figure 9.1: Taxonomy of computer systems [40].

The resources are shared between nodes, also called peers, and a peer can be either a server or a client. As defined by Wikipedia [14], P2P is divided into three categories:

Pure P2P: All peers can act as both clients and servers with no central server. All peers have the same responsibility with no central entity for management and coordination.

Hybrid P2P: Needs a central server which keeps information on peers and responds to nodes requesting information. Central entities are contacted by peers.

Mixed P2P: A combination of pure and hybrid P2P.

9.2 Mobile P2P networks

A preferable mobile P2P network is one based on a pure P2P model where the peers can discover each other without the assistance of a central node. This can be implemented in two ways [40]:

With infrastructure: Base-station-based networks like GSM or UMTS.

Without infrastructure: Networking using mobile ad hoc networks like WiFi, Bluetooth or Infrared.

Creating applications based on the pure P2P model, includes a significant amount of challenges. Forman and Zahorjan defines three categories of challenges for mobile computing which are highly relevant for mobile P2P systems in their article "The challenges for mobile computing" [29]:

Communication: Varying bandwidth, interference, security, users outside coverage area, frequent disconnections, delays and integration of heterogeneous network.

Mobility: Difficult to address devices and location-dependent.

Portability: Size, weight, limited battery power, exposed to weather, must operate in noisy surroundings.

Of the issues mentioned above, disconnections are probably the biggest challenge when designing P2P applications for mobile devices. The devices are constantly moving around and the connection may be lost at any time. This will often result in incomplete data transfers between peers, which must be accounted for in order to create solid and useful applications.

9.3 Mobile Ad Hoc Networks (MANET)

Mobile devices equipped with network technologies such as Bluetooth or Infrared can constitute a wireless personal area network (WPAN). According to Wikipedia, the definition of a personal area network (PAN) is: “A personal area network (PAN) is a computer network used for communication among computer devices (including telephones and personal digital assistants) close to one person. The devices may or may not belong to the person in question. The reach of a PAN is typically a few meters. PANs can be used for communication among the personal devices themselves (interpersonal communication), or for connecting to a higher level network and the Internet (an uplink). Personal area networks may be wired with computer buses such as USB and FireWire. A wireless personal area network (WPAN) can also be made possible with network technologies such as IrDA and Bluetooth”[24].

Several WPANs can make up a mobile ad hoc network (MANET). Wikipedia defines MANET as:

“A mobile ad-hoc network (MANET) is a self-configuring network of mobile routers (and associated hosts) connected by wireless links the union of which form an arbitrary topology. The routers are free to move randomly and organize themselves arbitrarily; thus, the network’s wireless topology may change rapidly and unpredictably. Such a network may operate in a standalone fashion, or may be connected to the larger Internet” [23].
Mobile phones use cellular technology relying on systems such as GSM and UMTS. These systems relies on a underlying infrastructure. Ad hoc mobile networks do not need such infrastructure, which has its advantages. “When Peer-to-Peer comes Face-to-Face: Collaborative Peer-to-Peer Computing in Mobile Ad hoc Networks” by Gerd Kortuem et al. lists three of these advantages [30]:

No infrastructure required: Since no infrastructure is required, the network can be deployed spontaneously when needed anywhere.

Self-organization: In a wired network, the topology is determined by the cables that connect the nodes to each other. In an MANET, the network is created as soon as two nodes are within each others PAN. This means that the network is continuously reconfigured as mobile devices enters and exits each other’s PANs.

Fault tolerance: Since there is no infrastructure, the mobile devices cannot fail because of a base station, which is the case in networks that rely on GSM communication. The only way a MANET can fail is if one node fails, but since the network is based on P2P communication, the network can easily be self-reconfigured.

Figure 9.2 shows the taxonomy of ad hoc networks, where the horizontal axis describes the range of each technology. They are Body(BAN), Personal(PAN), Local(LAN) and Wide Area Network(WAN). VMS is based on Bluetooth-technology, and is therefore used in PANs. The information shared between nodes in an ad hoc network may follow different paths depending on how many parties are involved. It can roughly be divided into two groups:

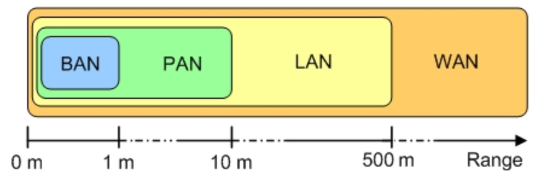


Figure 9.2: Ad hoc networks taxonomy [40].

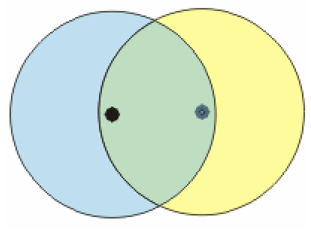


Figure 9.3: A Singlehop ad hoc network [40].

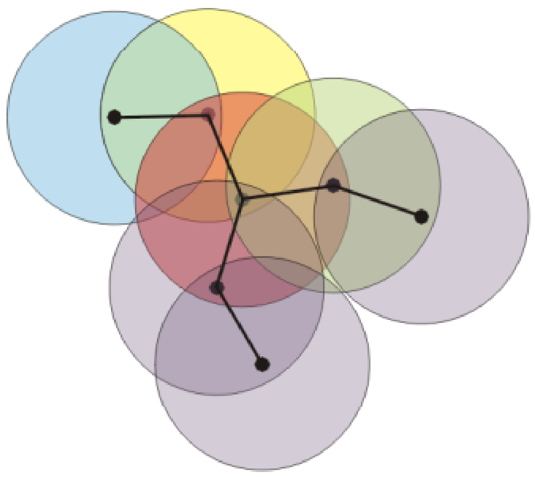


Figure 9.4: A Multihop ad hoc network [40].

Singlehop: Only two nodes are involved in a singlehop network, and the communication is performed directly between the two peers, as Figure 9.3 shows.

Multihop: A multihop configuration requires several nodes, where some nodes are out of direct contact with each other. The communication between these nodes are then forwarded by other intermediate nodes in the network. Figure 9.4 illustrates the multihop where the black lines are the communication paths.

9.4 Piconet

A piconet is a network of devices connected in an ad hoc fashion using Bluetooth technology. It is formed when atleast two devices connect (singlehop) and is converted to multihop when new nodes connect. It is sometimes called a PAN. A Bluetooth PAN can consist of up to eight devices, one of which must be a master and the other nodes act as slaves. It is important to keep in mind that the limit of eight devices means eight active devices, while a masternode theoretically can keep direct track of up to 255 parked devices. The master can "swap out" active slaves for parked slaves to manage piconets for situations that require a large number of connected devices, such as providing data services to people in high-population areas.

"Piconet" is a combination of the prefix "pico", meaning very small or one trillionth, and network.

9.5 Scatternet

A scatternet exists if a node is connected to a group of independent and non-synchronized piconets, so that they share at least one common Bluetooth device as illustrated in Figure 9.5. Bluetooth devices must have point-to-multipoint capability to engage in scatternet communication. There may be a maximum of 10 fully loaded piconets in a scatternet [16].

However, this means that a message from one node may have to be forwarded by several nodes, which raises security concerns. The problem with the current Bluetooth implementation is that it does not allow a node to function as both a master and a slave simultaneously, but sequentially.

9.6 P2P Technologies

There are several P2P technologies and protocols in use today. This section will describe a few of them.

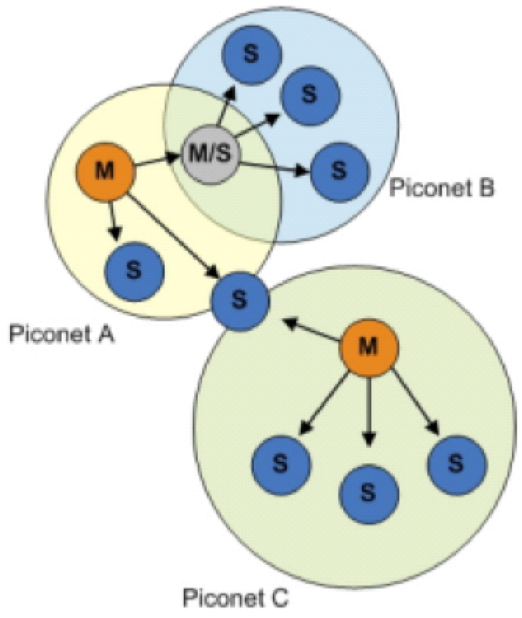


Figure 9.5: Piconets A, B and C together form a scatternet. m=master, s=slave, m/s=master and slave [40].

9.6.1 JXTA

Pronounced "juxta," JXTA is Sun Microsystem's set of open-source peer-to-peer networking protocols that allow any connected device on the network to communicate, including PC workstations and servers, cell phones and PDAs. Because it is based on protocols and not an API, JXTA works with any language, operating system, hardware and transport protocol. Virtually any network-capable device can be a JXTA peer. Because the underlying network does not have to be TCP/IP, JXTA applications can include Bluetooth-enabled mobile handsets as peers. JXTA is similar to Jini, but Jini networks require a Java Virtual Machine on every member device whereas JXTA does not.

JXTA provides the protocols for basic functions of peer-to-peer networking, such as creating, finding, joining, leaving and monitoring groups, talking to other groups and peers, and sharing content and services. The functions are performed by exchanging XML advertisements and messages between peers.

The name JXTA comes from the word juxtapose, meaning side-by-side [6].

9.6.2 Direct connect

Direct connect is a peer-to-peer file-sharing protocol. The most popular Windows client implementing this protocol currently is DC++. Direct connect clients connect to a central hub (usually on port 411) and can download files directly from one machine to another.

Hubs feature a list of clients or users connected to them. Users can search for files and download them from other clients, as well as chat with other users [25].

9.6.3 BitTorrent

BitTorrent (BT) is a peer-to-peer (P2P) communications protocol for file sharing. The protocol was designed in April 2001, implemented and first released July 2, 2001 by programmer Bram Cohen, and is now maintained by BitTorrent, Inc. BitTorrent is a method of distributing large amounts of data widely without the original distributor incurring the entire costs of hardware, hosting and bandwidth resources. Instead, when data is distributed using the BitTorrent protocol, recipients each supply data to newer recipients, reducing the cost and burden on any given individual source, providing redundancy against system problems, and reducing dependence upon the original distributor.

Usage of the protocol accounts for significant traffic on the Internet, but the precise amount has proven difficult to measure.

There are numerous compatible BitTorrent clients, written in a variety of programming languages, and running on a variety of computing platforms. More information about BitTorrent can be found in [47].

Chapter 10

Technology

This chapter describes the technologies used in the development of the VMS.

10.1 Mobile phones

A mobile or cellular telephone (commonly, “mobile phone” or “cell phone”) is a long-range, portable electronic device used for mobile communication. In addition to the standard voice function of a telephone, current mobile phones can support many additional services such as SMS for text messaging, email, packet switching for access to the Internet, and MMS for sending and receiving photos and video. Mobile phones also often have features beyond the ones previously described, including music (MP3) playback, memo recording, personal organizers, built-in cameras and camcorders, ringtones, games, radio, Push-to-Talk (PTT), infrared, Wifi and Bluetooth connectivity, call registers, ability to watch streaming video or download video for later viewing, video call and serve as a wireless modem for a PC.Most current mobile phones connect to a cellular network of base stations (cell sites), which is in turn interconnected to the public switched telephone network (PSTN) (the exception are satellite phones)[8]. Figure 10.1 shows some mobile phones.

10.2 Bluetooth

Peer2Me abstracts the network layer, making the communication semi-transparent to the developers. The current version of Peer2Me implements Bluetooth as its network layer. Bluetooth connections are automatic and consume less battery power. The big draws of Bluetooth are that it is wireless, inexpensive and automatic. The older Bluetooth 1.0 standard has a maximum transfer speed of 1 megabit per second (Mbps), while Bluetooth 2.0 can manage up to 3 Mbps. Bluetooth 2.0 is backward compatible with 1.0 devices [3].

Figure 10.1: Mobile phones [34].

10.2.1 Bluetooth Operation

Bluetooth networking utilizes low-power radio waves to transmit data. It communicates on a frequency of 2.45 gigahertz (actually between 2.402 GHz and 2.480 GHz, to be exact). This frequency band has been set aside by international agreement for the use of industrial, scientific and medical devices (ISM). Bluetooth development is controlled by The Bluetooth Special Interest Group (SIG)[10].

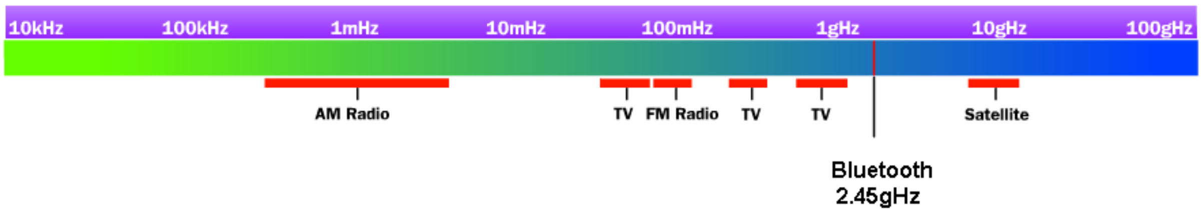


Figure 10.2: International radio frequency allocation [3].

Devices such as Baby monitors, garage-door openers and the newest generation of cordless phones all make use of frequencies in the ISM band. Bluetooth avoids interfering with other systems by sending out very weak signals of 1 milliwatt. The low power limits the range of a Bluetooth device to about 10 meters (32 feet), cutting the chances of interference.

Bluetooth can connect up to eight devices simultaneously with all of them in the same 10-meter (32-foot) radius. Bluetooth uses a technique called spread-spectrum frequency hopping that makes it rare for more than one device to be transmitting on the same frequency at the same time. In the case of Bluetooth, the transmitters change frequencies 1,600 times every second, making full use of a limited slice of the radio spectrum. Since every Bluetooth transmitter uses spread-spectrum transmitting automatically, it's unlikely that two transmitters will be on the same frequency at the same time [3].

The network created by bluetooth systems is called a Personal-area network (PAN) or

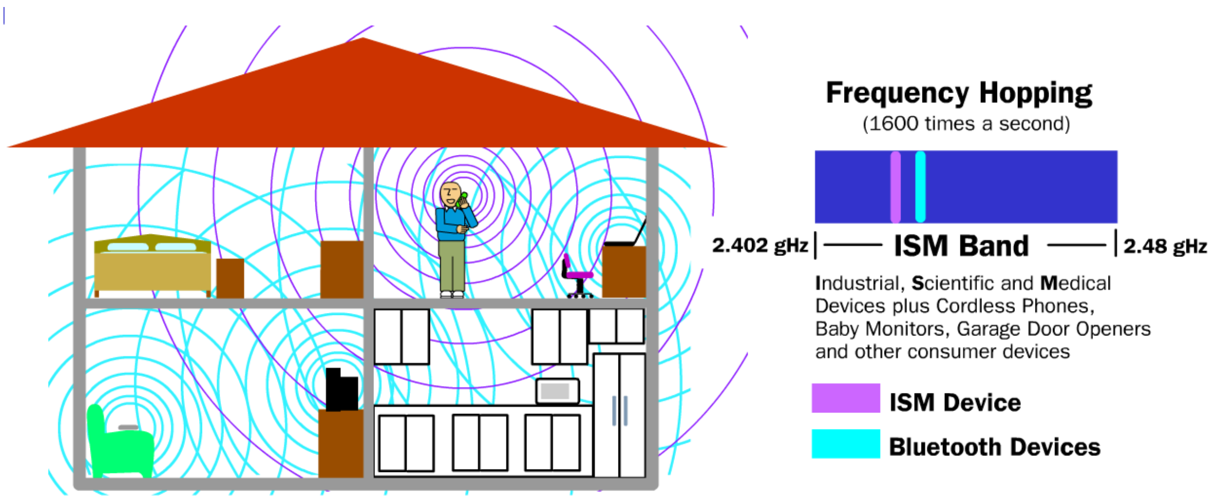

Frequency Hopping
(1600 times a second)
2.402 gHz
ISM Band
2.48 gHz
Industrial, Scientific and Medical Devices plus Cordless Phones, Baby Monitors, Garage Door Openers and other consumer devices
ISM Device
Bluetooth Devices

Figure 10.3: Bluetooth wireless piconet & frequency hopping [3].

piconet. Once the networks are established, the systems begin talking among themselves. Each piconet hops randomly through the available frequencies, so all of the piconets are completely separated from one another.

10.3 J2ME

J2ME is developed by Sun Microsystems and provides a complete application programming framework for all kinds of consumer and embedded devices, from phones to set-top boxes. These devices are characterized by features like mobility, limited memory, limited processing power, are battery powered, have small displays, and by limitations and variety with respect to input and output methods. J2ME specifications are divided into configurations specific to different device categories. Each configuration is specialized into a set of profiles.

10.3.1 Java platform

Java 2 Platform, Micro Edition (J2ME) is part of the Java 2 platform. While Java 2 Standard Edition (J2SE) targets desktop systems, and Java 2 Enterprise Edition (J2EE) targets the server backend applications, J2ME is a collection of APIs focusing on consumer and embedded devices, ranging from TV set-top boxes, telematics systems, residential gateways, to mobile phones and PDAs. Within each edition of the Java 2 platform, there are different Java Virtual Machine (JVM) implementations that are optimized for the type of systems they are targeted at. For example, the K Virtual Machine (KVM) is a JVM optimized for resource constrained devices, such as mobile phones and PDAs.

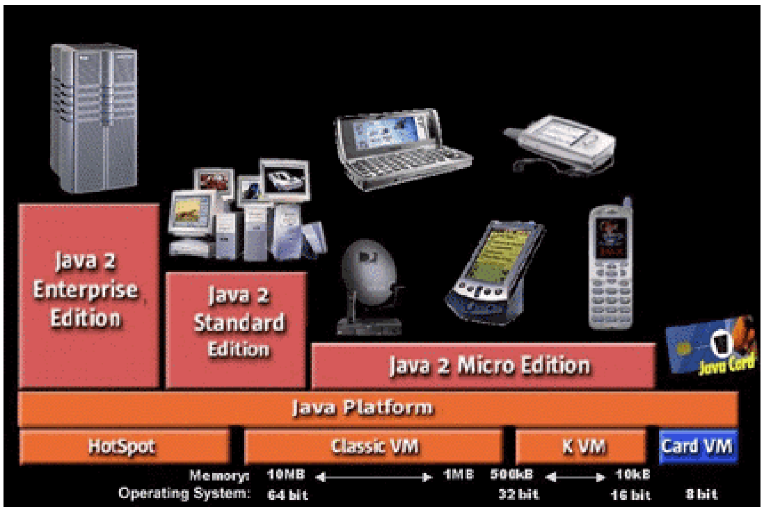

Java 2 Enterprise Edition
Java 2 Standard Edition
Java 2 Micro Edition
Java Card
Java Platform
HotSpot
Classic VM
K VM
Card VM
Memory: 10MB 1MB 500kB 10kB
Operating System: 64 bit 32 bit 16 bit 8 bit

Figure 10.4: Overview of Java 2 Platform [28].

10.3.2 Configurations

J2ME Configurations are specifications that describe a virtual machine and a base set of class libraries which provide the necessary APIs that can be used with a certain class of device. They provide the base functionality for a particular range of devices that share similar characteristics, such as network connectivity and memory footprint. A configuration, for example, might be designed for devices that have less than 512 KB of memory and an intermittent network connection. The virtual machine is either a full Java Virtual Machine (JVM), or some subset of the full JVM. The set of APIs is customarily a subset of the Java SE APIs. Currently, there are two Java ME configurations:

CDC - Connected Device Configuration. Used generally by devices that have more resources available such as PDAs and set-top boxes.

CLDC - Connected Limited Device Configuration. Designed for mobile phones, and are available in two versions, 1.0 (JSR82 30) and 1.1 (JSR82 139). 1.1 is a revised and backwards compatible version of 1.0. Version 1.0 is used in Peer2Me.

10.3.3 Profiles

J2ME Profiles complement a configuration by adding more specific APIs to make a complete runtime environment for running applications in a specific device category. It further defines the application life-cycle model, the user interface, persistent storage and access to device-specific properties. A widely adopted example is to combine CLDC with the Mobile Information Device Profile (MIDP) to provide a complete Java application environment for mobile phones and other devices with similar capabilities [5].

10.3.4 Optional packages

Optional packages are used to extend the functionality provided by CLDC or CDC and an associated profile(s). It provides standard API's for using technologies like database connectivity, wireless messaging, multimedia, 3D graphics, and web services. They can be implemented alongside virtually any combination of configurations and profiles. The development of the VMS will be done using the CLDC, MIDP and some optional packages.

Bluetooth: JSR82 82

The Java APIs for Bluetooth. This is a specification that standardizes a set of JAVA APIs to allow Java-enabled devices to integrate into a Bluetooth environment. The Peer2Me framework depends on this package and must be imported in the J2ME MIDlets that imports Peer2Me [40].

PIM: JSR82 75

The Java APIs for accessing PIM (Personal Information Management) data and the filesystem. This package is aimed at PDAs and high-end mobile phones. By having access to a deviceŠs PIM-information, an application can extract information from the phones calendar, address book etc. And having access to the filesystem is very useful and gives the opportunity to create powerful applications [40].

Mobile Media API (MMAPI): JSR 135

The Mobile Media API, JSR 135 , extends the functionality of the J2ME platform by providing audio, video, and other time-based multimedia support to resource-constrained devices. As a simple and lightweight optional package, it gives Java developers access to native multimedia services available on a given device.

The MMAPI is an optional package within the J2ME platform. While the main emphasis is on devices that implement profiles based on the Connected Limited Device Configuration (CLDC), the API design also aims at supporting devices that implement the Connected Device Configuration (CDC) and the profiles based on CDC [41].

10.3.5 What J2ME offers

J2ME complements technologies like WAP, cHTML and i-Mode. It allows you to run a MIDlet in a browser that supports WML or i-Mode. It also allows you to run a standalone J2ME application on a mobile phone, thus bringing all the notions of

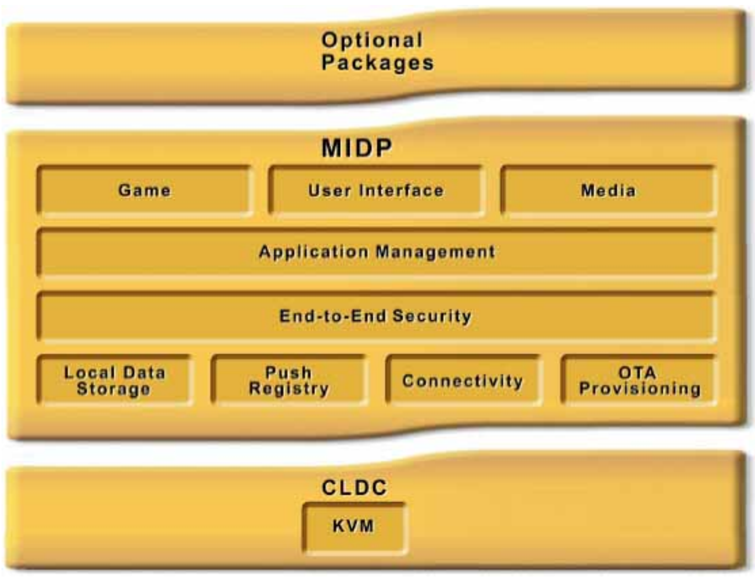


Figure 10.5: J2ME Architecture used in VMS [46].

Java on PC's to mobile phones. These applications could be games, navigation aids or applications interacting with databases. The idea of ubiquitous computing can be realized with this plateform [46].

Mark Weiser defines ubiquitous computing as: “First were mainframes, each shared by lots of people. Now we are in the personal computing era, person and machine staring uneasily at each other across the desktop. Next comes ubiquitous computing, or the age of calm technology, when technology recedes into the background of our lives” [19]. Alan Kay of Apple calls this ”Third Paradigm” computing.

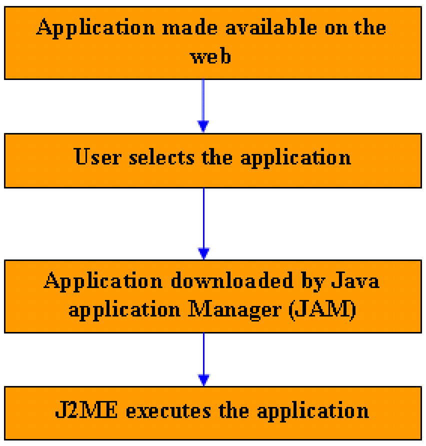

Application made available on the web
User selects the application
Application downloaded by Java application Manager (JAM)
J2ME executes the application

Figure 10.6: Downloading & running a J2ME application [46] .

Chapter 11

Peer2Me Background

This chapter gives an overview of the Peer2Me Framework and how to use it for mobile application development. The Peer2Me framework was developed as a result of a master thesis by Carl-Henrik Wolf Lund and Michael Sars Norum [35], and is a project related to the MOWAHS project Mobile Work Across heterogeneous Systems. The framework is currently in its second version having undergone rework by Kim Petter Saxlund and Tommy Bjørnsgård [40].

11.1 Introduction

The Peer2Me framework is a framework for developing applications on mobile phones and handheld devices, enabling users to communicate in an ad-hoc environment. The Peer2Me framework is based on the distributed system concept, Peer-to-peer (P2P). Peer2Me can be classified as a hybrid P2P model since implementation on mobile phones has shown that a device cannot function both as a server and a client at the same time using the Bluetooth technology. Furthermore, the Bluetooth API for mobile phones only supports connections for upto seven devices simultaneously. The creators of Peer2Me initially wanted it to support a pure P2P model, but this is currently very hard to accomplish.

11.2 Design

This section describes the design of Peer2Me.

11.2.1 Domain Concepts

In order to understand how the system works, some terms need to be explained:

Framework: The core entity of the framework. It manages the resources such as known peers and available network mediums. It is also the interface between the application and the rest of the system.

Node: A node is a peer; a mobile phone running the framework. A node must be either a slave or a master, not both.

Network: is an abstraction of the network layer. Is indirectly accessed through the framework instance.

Service: An application must be running a specific service in order for other peers to locate the application and connect to the respective peer.

Group: A collection of nodes running the same service. The group must have one master node.

Message: Messages are exchanged between the different nodes within a group in order to communicate.

Application: Software that uses the framework.

Figure 11.1 illustrates a conceptual model of how the Peer2Me framework works. A mobile phone is a node which runs an application with a specific service ID. This node then communicates with other nodes running the same service by messages which is sent using the framework. The network technology used is Bluetooth.

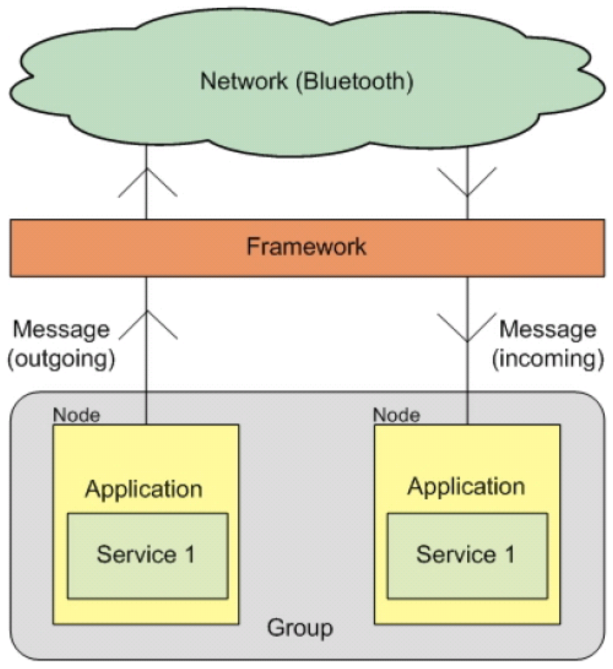


Figure 11.1: Domain concept of Peer2Me [39].

In order to make two or more nodes communicate, they have to be a "member" of the same group. The Peer2Me framework is built upon the core J2ME components, the CLDC (Connected, Limited Device Configuration) and the MIDP (Mobile Information Device Profile). The MIDP must be version 2.0 or later. Peer2Me consists of four main modules, the framework itself, the network interface, the Bluetooth module and the domain module. Figure 11.2 illustrates a layered architecture of Peer2Me, the J2ME core components and a Midlet.

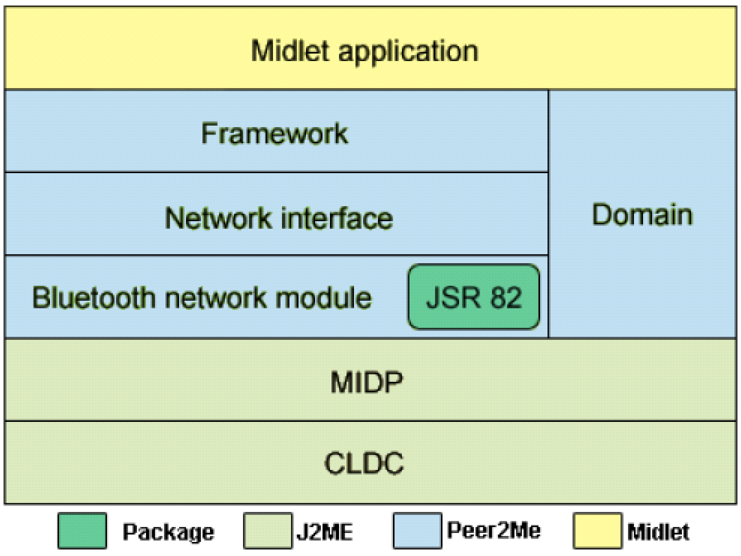


Figure 11.2: Layered architectural overview [39].

The generic network interface enables the developer to control the technology specific network modules. In the current version of Peer2Me, the network technology is Bluetooth. The idea is that new network technology modules can easily be added at a later point and the same network interface can be used in order to access it. The Bluetooth network module uses the JAWBT package called JSR82, see [15] for more information. The package is not a core J2ME component, but is optional. The Peer2Me framework is available as a jar-file. The classes in the jar-file is organized in a package structure. There are five main packages:

- no.ntnu.idi.mowahs.project.framework
- no.ntnu.idi.mowahs.project.bluetooth
- no.ntnu.idi.mowahs.project.domain
- no.ntnu.idi.mowahs.project.network
- no.ntnu.idi.mowahs.project.util

The usage area of the first four packages are illustrated in Figure 11.1 and Figure 11.2. The util package contains "persistence" classes that can be used to access the recordstore on mobile phones. This package is only used by MIDlets that requires access to the recordstore[1]. More information about the Peer2Me framework can be found in [40].

[1]A J2ME compatible mobile phone can store application data in a recordstore. The recordstore is a persistent storage which MIDlets can store and retrieve data at any time

Chapter 12

State of the Art

This chapter describes technologies/solutions similar to the voice messaging system that we are working on. It also states the differences of those technologies/solutions with respect to the VMS. It is important to note that although the systems described in this section provide a functionality similar to the VMS, but are completely different in its technology and architecture. We use a peer-to-peer approach for voice messaging and we did not manage to find any other project similar to our approach.

12.1 Telenor's Bubble Message

The Bubble message is a voice sms offered by Telenor Pakistan, that allows you to send voice messages [1]. Once the Bubble Message is sent, the recipient receives a notification via SMS, with the following information: Sender mobile number along with the number of messages sent by that sender, date and time of the Bubble Message deposit. To retrieve a Bubble Message (Any Telenor or djuice number) *0* is dialed to retrieve new message and *1* to retrieve old messages.

It is different from the VMS in the following ways:

Architecture: Bubble message utilizes the infrastructure of GSM for its operation. It is based on the client server architecture. The VMS is P2P and does not require additional infrastructure hence can be deployed spontaneously when needed anywhere.

Cost: Bubble message is a commercial service and is not for free. Since VMS is based on PAN technologies, and does not require additional infrastructure it does not cost.

12.2 DiGi's BubbleTalk

Malaysian mobile phone operator DiGi Telecommunications also provides a Short Voice Message Service (SVMS)[9]. It is like an SMS with voice instead of text and provides about 30 seconds to record you voice-SMS.

Its differences with the VMS are similar to those described in Section 12.1.

12.3 HP OpenCall Voice Short Message Service (SMS) solution

The HP OpenCall Voice short message service solution is also a commercial product. It provides multimedia messaging solutions to service providers and also to enterprises [4]. Its different from the VMS in the following ways:

Architecture: The HP OpenCall Voice utilizes the the HP OpenCall Media Platform for its operation. It is based on the client server architecture. The VMS is P2P and does not require additional infrastructure.

Cost: Bubble message is a comercial service and is not for free. Since VMS is based on PAN technologies, and does not require additional infrastructure it does not cost.

12.4 Voicemail

Voice mail is a common service offered by almost all of the telephony service providers. If a person is using his phone or doesn't answer it, the call is forwarded to a machine - the voicemail system [20]. The VMS is a totally different concept than the voicemail. VMS enables end users to leave a message without interrupting the receiver with a phone call. Moreover the service is free to use.

Part IV

Own Contribution: Voice Messaging System VMS

Chapter 13

Description

13.1 Introduction of VMS

The initial work on the Voice Messaging System (VMS) was started as a Depth study by Hassan [42]. This work is in continuation with and builds on that project.

The voice messaging system (VMS) is a mobile phone application that allows end users to send quick voice recordings to family, friends, and colleagues from their mobile phones without using their mobile service providers network. It is an application that uses bluetooth piconet (Section 9.4) and hence is free to use. It provides users with an intutive and much more personal alternative to typing short text messages (SMS) in a user friendly way.

13.2 User Benefits

The VMS has a number of benefits for the end users. These are described below:

- Recording a spoken message is less trouble and faster than by typing a text SMS.
- Users can send messages in any language they prefer instead of typing their message with the phone keypad that might not support required characters.
- It is much more easier to comprehend a voice message as compared to a text SMS.
- Voice adds an emotional and personal dimension to messaging.
- It is very useful for disabled users, having difficulty with typing text on mobile keypads.
- It does not require the installation of network equipment/infrastructure, but uses the available bluetooth on mobile phones. The system can be deployed spontaneously when needed anywhere.

- Last but not the least, It is free to use.

13.3 Limitations

As mentioned before, the VMS is based on a peer2peer network. Thus it limits communication to peers only accessible on the network. However, this network can be extended, by pairing the Bluetooth piconets with the Internet, but this will not be the scope of this project. It can be a useful extention for future work.

Chapter 14

Requirements

14.1 Functional Requirements

This section describes the functional requirements of the VMS application. Functional requirements are a set of instructions reflecting the functionality which must be implemented in the application. The requirements are presented in Table 14.1.

The voice messaging system (VMS) is an application that can be installed on Java enabled mobile phones with bluetooth connectivity. It allows its users to send quick voice recordings (voice messages) to one another. In order to send a message to a friend, the user records his/her voice message using the VMS application and selects the reciepent from the available list of users in range. The recording can be heard/changed before sending. After this, the VMS application allows the user to send his/her message to the selected user.

A user after recieving a voice message is informed of its arrival by the VMS application. After recieving a voice message, it can be heard by using the appropriate option in the application. The VMS also stores all incoming and outgoing voice messages for future reference.

14.2 Quality Requirements

The quality requirements, sometimes refered to as non-functional requirements is an important consideration when designing applications. They are often as important as the functional requirements. This section describes the quality requirements that the VMS application should have. The reader should distinguish between the the follwing roles mentioned in this section:

The developer - The developer who is responsible for changes and modifications to

FR1	The VMS should be able to send and receive voice messages to other peers/nodes.
FR2	The VMS should provide an interface for recording voice messages.
FR3	The VMS should provide an interface to playback received messages.
FR4	The VMS should use the Peer2Me framework.
FR5	The VMS should store incomming voice messages.
FR6	The VMS should alert the user of the arrival of a new message .
FR7	The VMS should provide functionality to delete messages.
FR8	The VMS should provide functionality to forward a message to another node.
FR9	The VMS should provide an inbox, where it is possible to view all received messages.
FR10	The VMS should provide an outbox, where it is possible to view all sent messages.

Table 14.1: Functional requirements of VMS

M1 - Make changes to an existing component of VMS	
Source of stimulus	The developer
Stimulus	The developer wants to make changes in an existing component of the VMS
Environment	Design time
Artefact	A component of the VMS application
Response	Changes are made to that component only/ at one place.
Response Measure	Changes made in 1 day

Table 14.2: Modifiability scenario M1

the application.

The user - The person who uses the application developed by the application developer.

14.2.1 Modifiability

Modifiability deals with changes to the system. The design of the VMS should allow for easy future modifications and/or additional modules. In order to satisfy this requirement, the application will have clearly defined modules based on their functionality and semantic coherence. Table 14.2 presents a modifiability scenario.

14.2.2 Usability

Usability is associated with the end-user of the VMS. It is concerned with how easy it is for the end-user to perform a certain task and how the system displays information

U1 - Provide assistance to the end-user	
Source of stimulus	The end-user
Stimulus	The end-user wants to send a voice message
Environment	Runtime
Artefact	The VMS application
Response	The application provides step by step instructions to the end-user
Response Measure	The end-user succeeds in sending the message 99% of the time without help

Table 14.3: Usability scenario U1

T1 - Test harness	
Source of stimulus	The developer
Stimulus	The developer wants to examine variable values in components
Environment	Debug mode
Artefact	The application Code
Response	The code provides special testing interfaces to capture variable values inside components
Response Measure	The testing interfaces provide variable values 99.9% of the time

Table 14.4: Testablity scenario T1

to the user. The VMS will be designed to provide basic instructions to the end-user with a well defined user interface at runtime. Table 14.3 presents a usability scenario.

14.2.3 Testability

Testability deals with finding bugs and faults in the system. An architecture that can be easily tested for faults saves a lot of time. This requirement will be met by providing special testing interfaces inside the various modules to capture inner variable values for testing purposes. Table 14.4 presents a Testablity scenario.

14.2.4 Availability

Since the VMS operates through a wireless Bluetooth network (see Section 9.4), availabilty is crucial for its successful operation. The VMS will make use of the availabilty tactics provided by the Peer2Me framework. Two availability scenario's are described in Table 14.5 and Table 14.6.

A1 - Disconnection of node	
Source of stimulus	A lost node
Stimulus	A node disconnects from the network because of a fault or under normal conditions.
Environment	Runtime
Artefact	VMS application
Response	The application becomes aware of the node disconnection
Response Measure	Disconnection registered with in 20 seconds

Table 14.5: Availability scenario A1

A2 - A new node arrives	
Source of stimulus	A new node
Stimulus	A new node joins the network
Environment	Runtime
Artefact	Network module
Response	The application becomes aware of the new node
Response Measure	New node registered with in 20 seconds

Table 14.6: Availability scenario A2

Chapter 15

Software Architecture

This chapter describes the software architecture of the VMS application. It is possible to make such an application without paying any attention to the architecture, but doing so results in a poor, low-quality, non-flexible and over-complex application. Software architecture ensures that the system meets the quality requirements set out by the different stakeholders of a system.

15.1 High level Architechture

Figure 15.1 shows the high level architectural overview of the VMS application. An architectural pattern is concerened with the construction context of the whole system, rather than just some part of the system. It is based on a layered architectural pattern. The five layers include the VMS application, the Peer2Me framework, the J2ME, the operating system of the mobile device and the hardware. Since the VMS will use functionality provided by the Peer2Me framework, it lies on top of it.

The layered architectural pattern provides a good abstraction and low maintenace. Another benefit of this approach is that it breaks complex problems into smaller, more manageable pieces. For instance, the VMS application will make use of the networking capabilities of the Peer2Me framework, to find other nodes and communicate via bluetooth.

One of the goals of this project was to explore the feasibility of the Peer2Me framework for the development of voice/mp3 messaging/file transfer applications like the VMS. For this purpose, it is important to test the functionalities provided by the Peer2Me framework with the VMS. This has also been a reason to use the layered architectural approach.

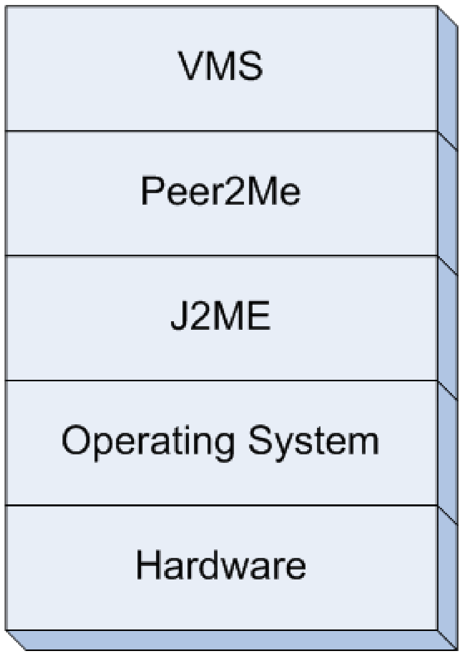


Figure 15.1: High level Architectural overview of the VMS.

Chapter 16

High Level Design

This chapter presents a logical overview of the VMS. It then describes the different components of the design.

16.1 Logical Structure of the VMS

The logical structure of the VMS that was initially designed in the Depth project by Hassan [42] was slightly changed to realize a better compostion of components and hence a better implemention. It is shown in Figure 16.1. The model-view-controller (MVC) architectural pattern (see Section 8.1.1) was also used in the design of the VMS. It contains nine main components namely, the VMSMidlet,VmsView, NewVms, media player/recorder, Inbox, Outbox, communication manager, the Peer2Me framework, Inbox Store and Outbox Store. Each of these components is described next.

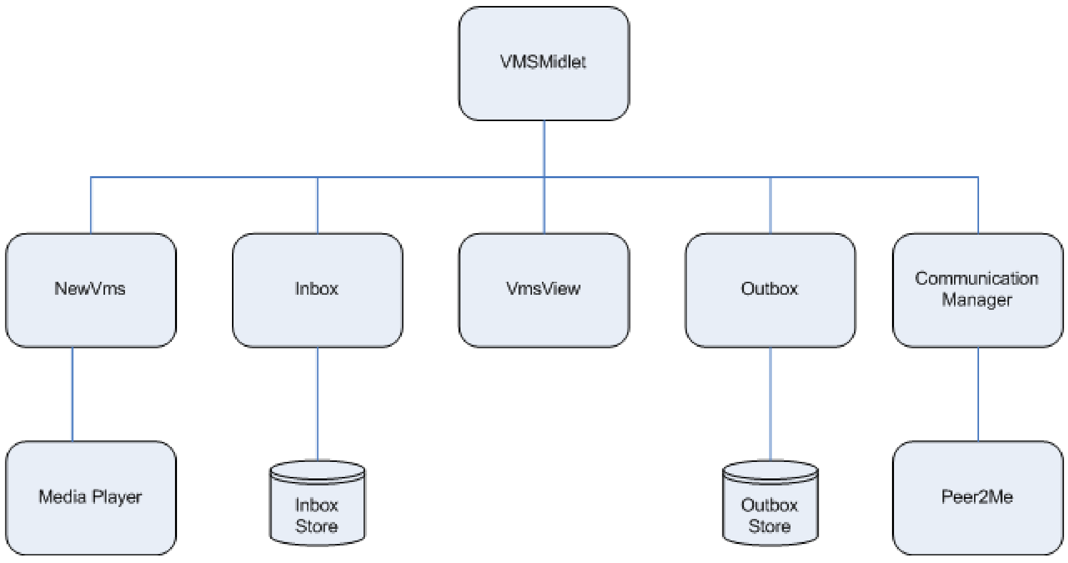


Figure 16.1: Logical structure of the VMS.

16.1.1 VMSMidlet

The VMS Midlet is the top entity of the application. It works as the main controller of the system. This was done according to the model-view-controller (MVC) architectural pattern (see Section 8.1.1) used for the system. The midlet starts from this class. Functioning as the controller, the VMSMidlet class handles most of the user generated events on the phone.

16.1.2 VmsView

The VmsView is part of the “View” of the MVC architectural pattern. It displays a list of three options to the user for navigation to the inbox, outbox or new voice message screen. It is the simplest class of the VMS application.

16.1.3 NewVms

The newVms is also a part of the “View” of the MVC architectural pattern. It presents the user with the visual interface to record real time voice and for its playback. It also provides the interface for searching peers and sending the new voice message.

16.1.4 Media player

The media player is part of the “Model” of the MVC architectural pattern. It allows the capture of real time audio from the phone microphone and its playback.

16.1.5 InboxView

The InboxView is part of the “View” of the MVC architectural pattern. It provides the visual interface to the inbox of the VMS System. It also provides methods for opening, deleting and forwarding voice messages.

16.1.6 Inbox Store

The Inbox Store is part of the “Model” of the MVC architectural pattern. It is designed on the record management system (RMS) provided in MIDP and stores all the incomming voice messages.

16.1.7 OutboxView

The OuboxView is part of the “View” of the MVC architectural pattern. It provides the visual interface to the outbox of the VMS System. It also provides methods for opening and deleting voice messages.

16.1.8 Outbox Store

The Outbox Store is part of the “Model” of the MVC architectural pattern. It is designed on the record management system (RMS) provided in MIDP and stores all the out-going voice messages.

16.1.9 Communcation manager

The communication manager is also part of the “Model” of the MVC architectural pattern. It provides interfaces to the Peer2Me framework (Chapter 11). It handles the connection between the VMS application and the Peer2Me framework such as sending and recieving of messages.

16.1.10 Peer2Me

The Peer2Me framework is an important component of the VMS application. It provides the basic functionalities of networking and communication to the application. The application also uses the message format provided by the Peer2Me framework and extends it with audio properties.

16.2 High level general process scenarios

This section presents how data is passed between the VMS application and the different layers in the Peer2Me framework in some typical scenarios. This section also presents how nodes change roles depending on which node initiates a transfer of a message.

The VMS interacts with the framework through the communication manager and the Framework class. When the Peer2Me framework wants to send data to the VMS application, it does this through the FrameworkSubscriber. Figure 16.2 shows how the Peer2Me is initialized when the VMS application calls the method initialize() through its communication manager on the Framework class.

After the Peer2Me framework has been initialized, the VMS can start a search for other nodes. This is why it is initialized by the communication manager as the application

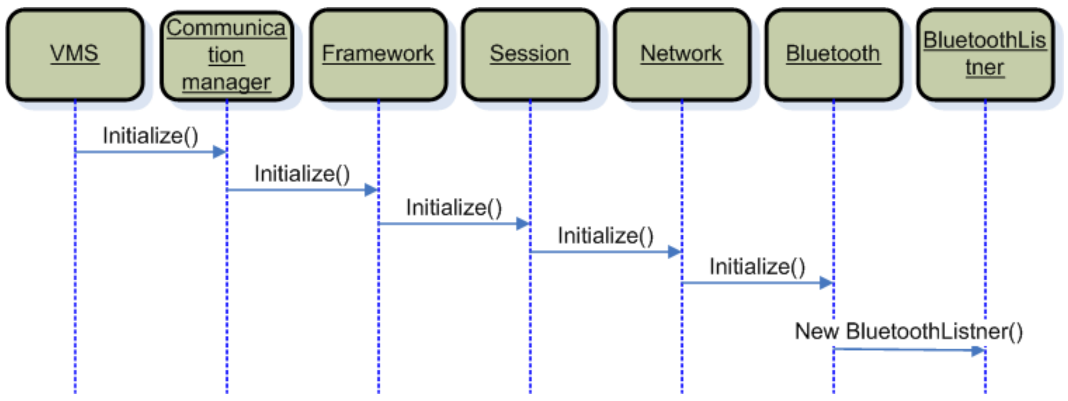


Figure 16.2: Initialization of the Peer2Me framework.

starts. Figure 16.3 shows how the framework initiates a search through the different layers and how the framework alerts the VMS when a node is found.

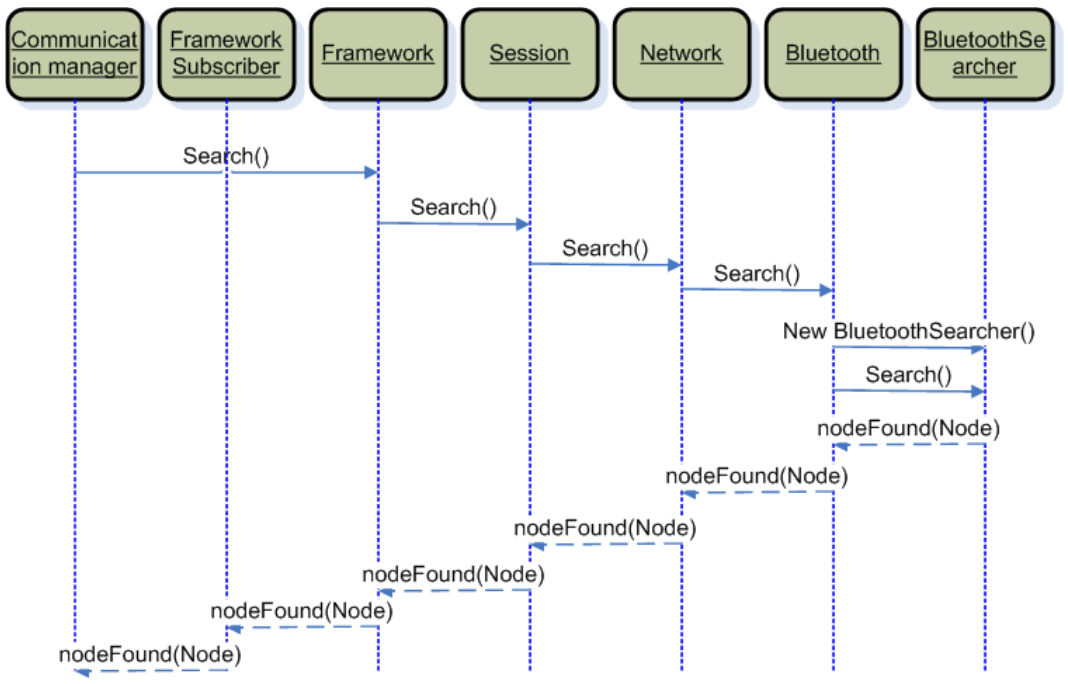


Figure 16.3: Search for other nodes.

After a search has completed, a node can send a message to another node. Since Peer2Me implements a pure peer-to-peer Bluetooth network, nodes are independant of a central node and must be able to change “roles” depending on the situation. Figure 16.4 shows three nodes which are all aware about each other, but none of them have been assigned either a master or a slave role.

If node A wants to send a message to node B, node A will take the role as master and node B becomes a slave, see Figure 16.5.

Later, if node B wants to send a message to node C, node B will become the master of

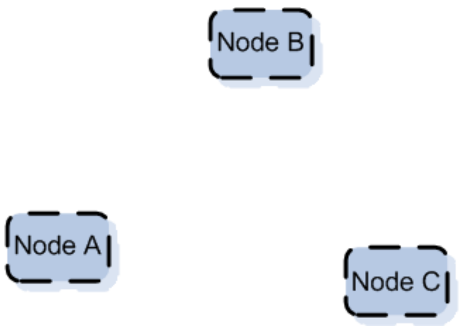


Figure 16.4: None of the nodes has been assigned a role [40].

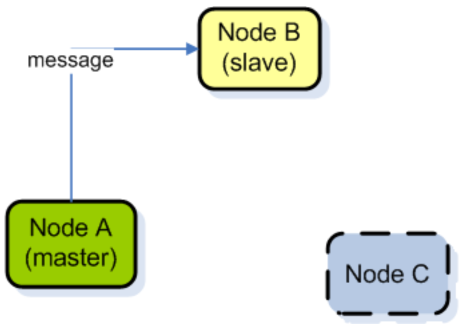


Figure 16.5: Node A sends a message to node B [40].

that connection. The transfer of a message from B to C is illustrated in Figure 16.6. More information on the Peer2Me framework can be found in [40].

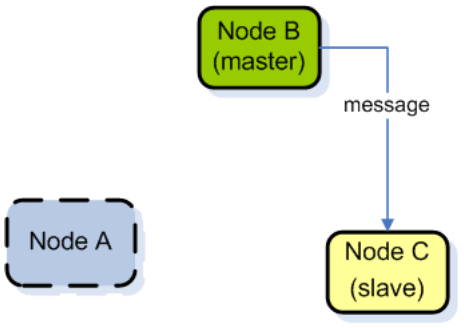


Figure 16.6: Node B sends a message to node C [40].

Chapter 17

Implementation

This chapter describes the implementation process of the VMS application. It also describes the methodology used to realize the VMS application.

17.1 Coding

The coding of the VMS application was carried out in MIDP J2ME (Section 10.3). Eclipse SDK (Section 8.2.1) was used as the programming environment. The implementation was carried out in three main phases for the development of the user interface, controller and model of the VMS application. The system modules shown previously in Figure 16.1 were implemented as separate classes. The application was divided into seven different classes. The VmsView, NewVmsView, InboxView and OutboxView classes constituted the View of the MVC architectural pattern (Section 8.1.1). The Communicator and PlayerRecoder classes formed the model. The midlet class was used as the main contoller for the application. Since the VMS was being designed for use on mobile phones, there were limitations. These limitations are especially noticeable in the areas of the user interface and available memory. For the user interface the main challenges were the small display size of the mobile device. Some of the programming techniques that were used to ensure high performance and efficient use of resources are listed below which will prove helpful in the development of any mobile application.

Conserve memory - Due to the small memory budget available, the size of the application data of the VMS was kept to a minimal. One such example is the limited recording of audio, which can be increased on devices with more memory. Another technique was to use local variables where ever possible and minimal use of objects to improve performance.

Do not overload the processor - Processor speeds for small devices are significantly less than those for PCs. While writing the VMS application, unnecessary load was avoided such as not changing the audio format.

Do not hold on to resources when not needed - Resources such as RMS,files, network connections, and so on should not be kept up when they are not in use. Necessary cleanup operations (such as assigning null value to objects when they are no longer needed)were performed rather than depending on the garbage collector or the host environment.

Use local variables - When loops are used, class members are called repeatedly. The execution of the program can be made faster by using local variables instead of class members. The value of the class member can be assigned to a temporary variable on the stack. This value can then be used in place of the class member. The same applies to arrays also.

17.1.1 UML class diagram of the VMS

Figure 17.1 shows the UML class diagram of the VMS system. As described before, it contains seven classes, an image folder containing the images used in the application and the Peer2Me framework package. A detailed description of these classes is given next.

UML class diagram of VMSMidlet.java

The UML class diagram of VMSMidlet.java is shown in Figure 17.2. This class functions as the main controller for the VMS application. Most of the user generated actions are handled from here. It also extends the Midlet class of MIDP. The source code for this class can be found in Appendix A.1.

UML class diagram of NewVmsView.java

The UML class diagram of NewVms.java is shown in Figure 17.3. This class provides the user interface for recording a new voice message. It uses a PlayerRecorder object for recording and playback of audio. It also provides interfaces for forwarding and replying a voice message. The source code for this class can be found in Appendix A.2.

UML class diagram of PlayerRecorder.java

The UML class diagram of PlayerRecorder.java is shown in Figure 17.4. It provides the basic functionalities of recording and playback of audio. The record function is executed in a separate thread to avoid locking the main thread running the application. The source code for this class can be found in Appendix A.4.

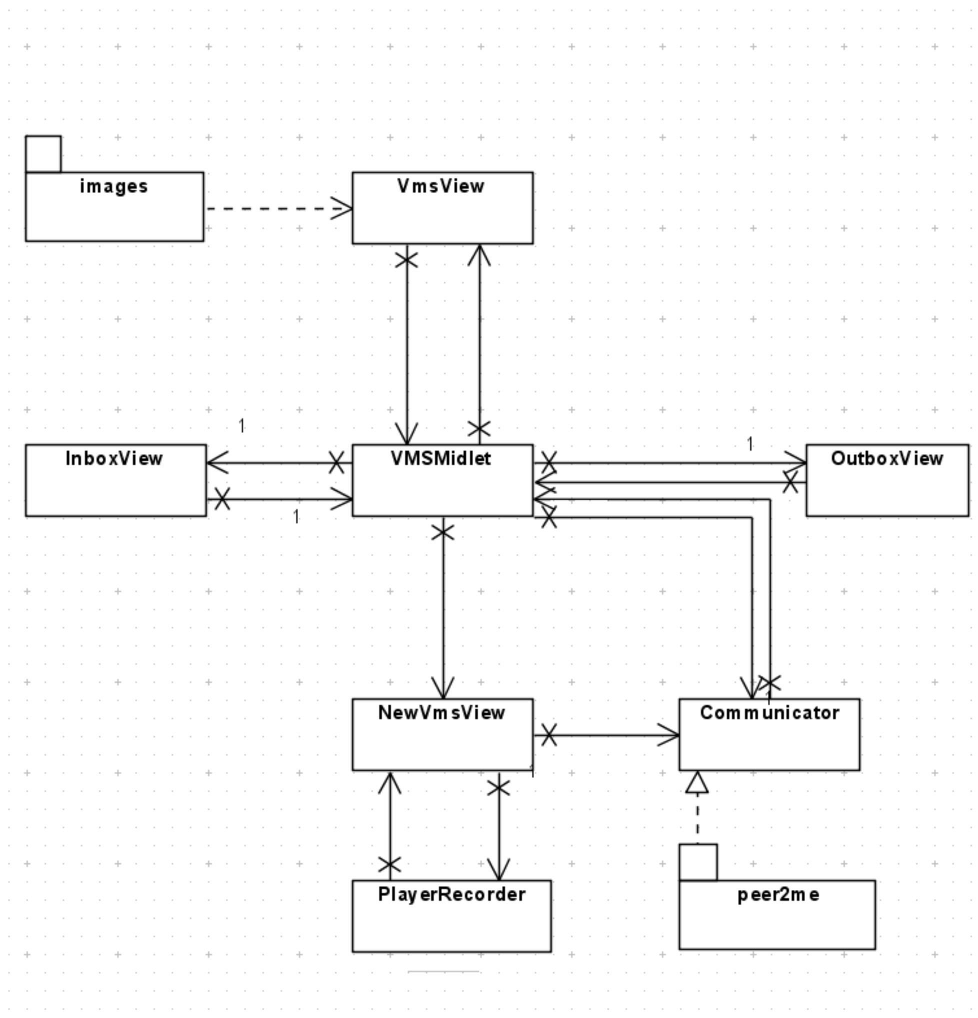
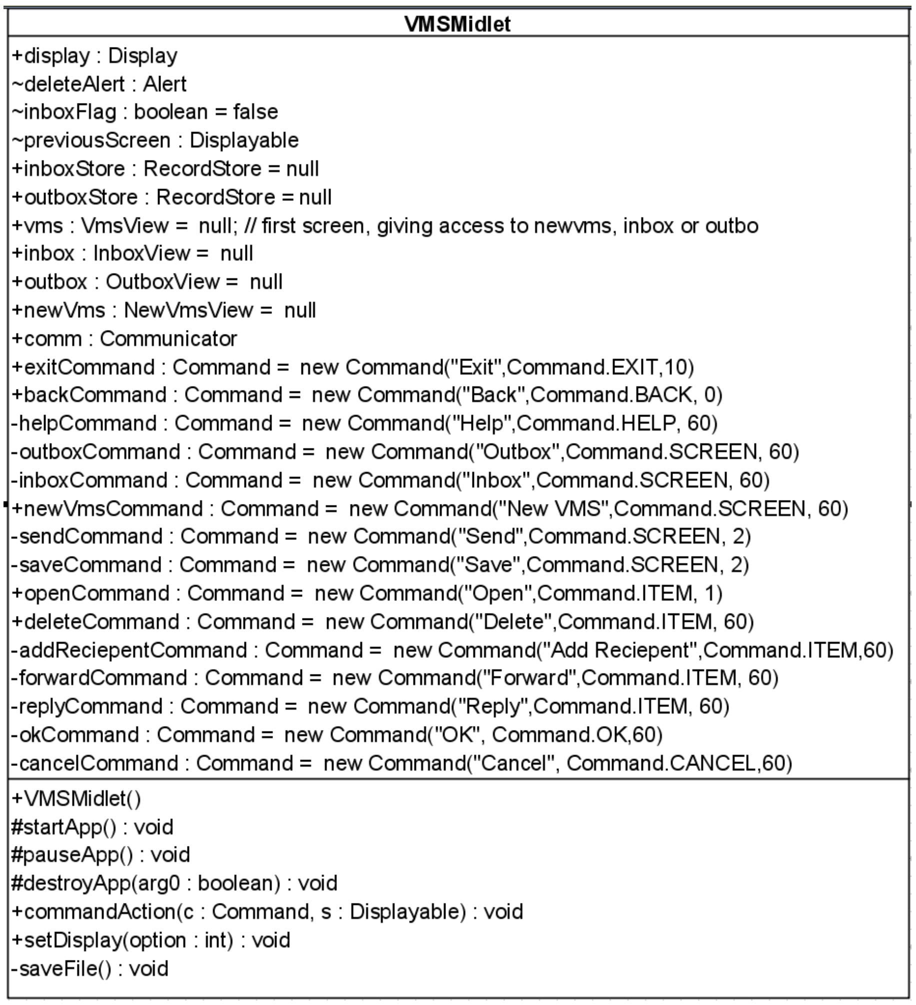
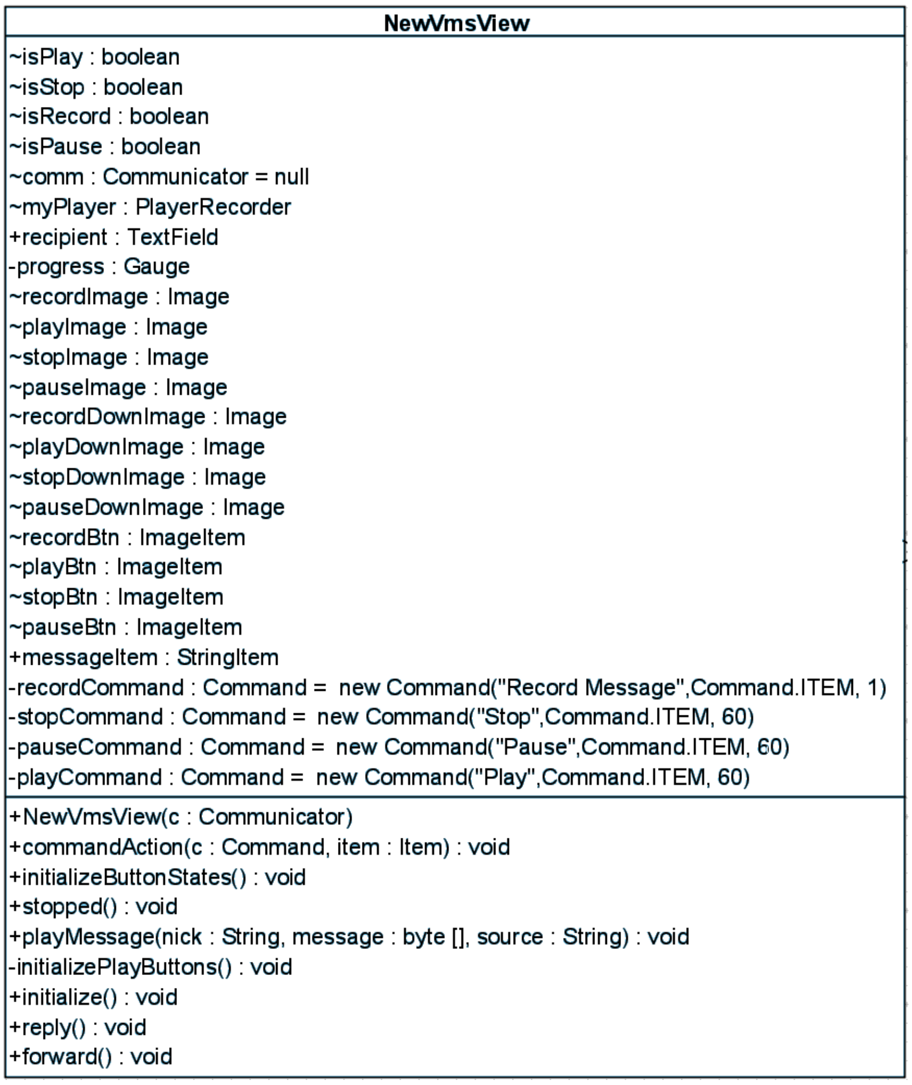


Figure 17.1: UML class diagram of the VMS system



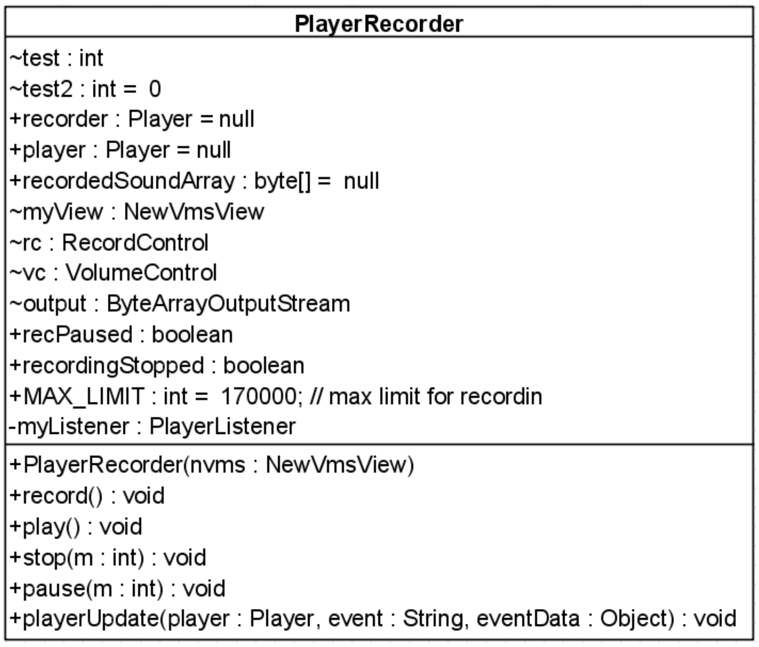
```
VMSMidlet
+display : Display
~deleteAlert : Alert
~inboxFlag : boolean = false
~previousScreen : Displayable
+inboxStore : RecordStore = null
+outboxStore : RecordStore = null
+vms : VmsView = null; // first screen, giving access to newvms, inbox or outbo
+inbox : InboxView = null
+outbox : OutboxView = null
+newVms : NewVmsView = null
+comm : Communicator
+exitCommand : Command = new Command("Exit",Command.EXIT,10)
+backCommand : Command = new Command("Back",Command.BACK, 0)
-helpCommand : Command = new Command("Help",Command.HELP, 60)
-outboxCommand : Command = new Command("Outbox",Command.SCREEN, 60)
-inboxCommand : Command = new Command("Inbox",Command.SCREEN, 60)
+newVmsCommand : Command = new Command("New VMS",Command.SCREEN, 60)
-sendCommand : Command = new Command("Send",Command.SCREEN, 2)
-saveCommand : Command = new Command("Save",Command.SCREEN, 2)
+openCommand : Command = new Command("Open",Command.ITEM, 1)
+deleteCommand : Command = new Command("Delete",Command.ITEM, 60)
-addReciepentCommand : Command = new Command("Add Reciepent",Command.ITEM,60)
-forwardCommand : Command = new Command("Forward",Command.ITEM, 60)
-replyCommand : Command = new Command("Reply",Command.ITEM, 60)
-okCommand : Command = new Command("OK", Command.OK,60)
-cancelCommand : Command = new Command("Cancel", Command.CANCEL,60)
+VMSMidlet()
#startApp() : void
#pauseApp() : void
#destroyApp(arg0 : boolean) : void
+commandAction(c : Command, s : Displayable) : void
+setDisplay(option : int) : void
-saveFile() : void
```

Figure 17.2: UML class diagram of VMSMidlet.java



```
NewVmsView
~isPlay : boolean
~isStop : boolean
~isRecord : boolean
~isPause : boolean
~comm : Communicator = null
~myPlayer : PlayerRecorder
+recipient : TextField
-progress : Gauge
~recordImage : Image
~playImage : Image
~stopImage : Image
~pauseImage : Image
~recordDownImage : Image
~playDownImage : Image
~stopDownImage : Image
~pauseDownImage : Image
~recordBtn : ImageItem
~playBtn : ImageItem
~stopBtn : ImageItem
~pauseBtn : ImageItem
+messageItem : StringItem
-recordCommand : Command =  new Command("Record Message",Command.ITEM, 1)
-stopCommand : Command =  new Command("Stop",Command.ITEM, 60)
-pauseCommand : Command =  new Command("Pause",Command.ITEM, 60)
-playCommand : Command =  new Command("Play",Command.ITEM, 60)
+NewVmsView(c : Communicator)
+commandAction(c : Command, item : Item) : void
+initializeButtonStates() : void
+stopped() : void
+playMessage(nick : String, message : byte [], source : String) : void
-initializePlayButtons() : void
+initialize() : void
+reply() : void
+forward() : void
```

Figure 17.3: UML class diagram of NewVmsView.java

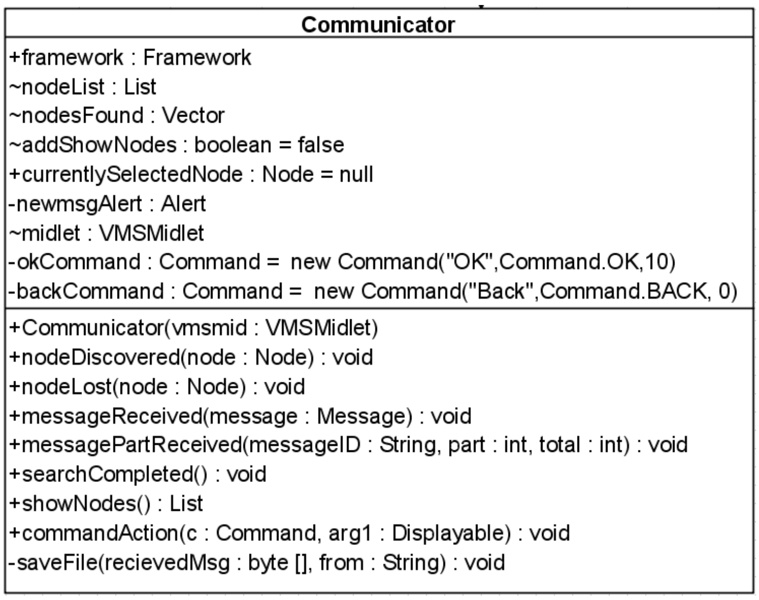


```
PlayerRecorder
~test : int
~test2 : int =  0
+recorder : Player = null
+player : Player = null
+recordedSoundArray : byte[] =  null
~myView : NewVmsView
~rc : RecordControl
~vc : VolumeControl
~output : ByteArrayOutputStream
+recPaused : boolean
+recordingStopped : boolean
+MAX_LIMIT : int =  170000; // max limit for recordin
-myListener : PlayerListener
+PlayerRecorder(nvms : NewVmsView)
+record() : void
+play() : void
+stop(m : int) : void
+pause(m : int) : void
+playerUpdate(player : Player, event : String, eventData : Object) : void
```

Figure 17.4: UML class diagram of PlayerRecorder.java

UML class diagram of Communicator.java

The UML class diagram of Communicator.java is shown in Figure 17.5. This class provides interfaces to connect to the Peer2Me framework. This class implements the FrameworkSubscriber interface of the Peer2Me framework and is used by the VMS application to send and receive voice messages. Other nodes are also searched through the Communicator class. The source code for this class can be found in Appendix A.7.



```
Communicator
+framework : Framework
~nodeList : List
~nodesFound : Vector
~addShowNodes : boolean = false
+currentlySelectedNode : Node = null
-newmsgAlert : Alert
~midlet : VMSMidlet
-okCommand : Command =  new Command("OK",Command.OK,10)
-backCommand : Command =  new Command("Back",Command.BACK, 0)
+Communicator(vmsmid : VMSMidlet)
+nodeDiscovered(node : Node) : void
+nodeLost(node : Node) : void
+messageReceived(message : Message) : void
+messagePartReceived(messageID : String, part : int, total : int) : void
+searchCompleted() : void
+showNodes() : List
+commandAction(c : Command, arg1 : Displayable) : void
-saveFile(recievedMsg : byte [], from : String) : void
```

Figure 17.5: UML class diagram of Communicator.java

UML class diagram of VmsView.java

The UML class diagram of VmsView.java is shown in Figure 17.6. This is a simple class which provides navigation to the inbox, outbox and the new VMS message screen. The source code for this class can be found in Appendix A.5.

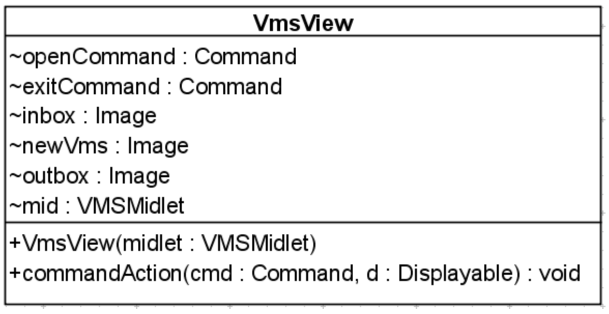


Figure 17.6: UML class diagram of VmsView.java

UML class diagram of InboxView.java

The UML class diagram of InboxView.java is shown in Figure 17.7. It displays the received messages in a list format. It also provides functionalities for opening and deleting messages. The source code for this class can be found in Appendix A.3.

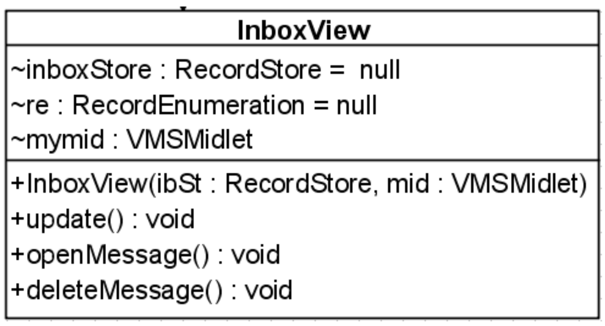

InboxView
~inboxStore : RecordStore = null
~re : RecordEnumeration = null
~mymid : VMSMidlet
+InboxView(ibSt : RecordStore, mid : VMSMidlet)
+update() : void
+openMessage() : void
+deleteMessage() : void

Figure 17.7: UML class diagram of InboxView.java

UML class diagram of OutboxView.java

The UML class diagram of OutboxView.java is shown in Figure 17.8. It displays the sent messages in a list format. It also provides functionalities for opening and deleting messages. The source code for this class can be found in Appendix A.6.

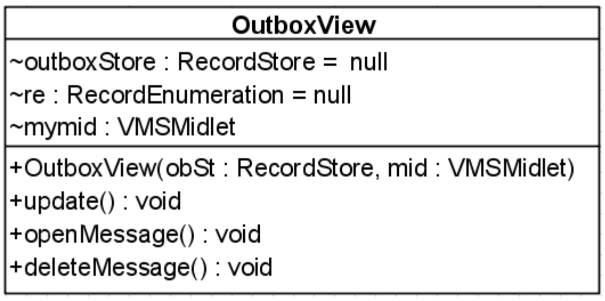

OutboxView
~outboxStore : RecordStore = null
~re : RecordEnumeration = null
~mymid : VMSMidlet
+OutboxView(obSt : RecordStore, mid : VMSMidlet)
+update() : void
+openMessage() : void
+deleteMessage() : void

Figure 17.8: UML class diagram of OutboxView.java

17.2 Implementing the User Interface

The user interface (UI) of the VMS application was implemented to be user friendly. Different components were used to make the UI. The following are some of the most important ones:

1. An Alert acts as a screen to provide information to the user about an exceptional condition or error.
2. A Form acts as a container for the other UI elements.
3. A List provides a list of choices.
4. A StringItem acts as a display-only string.
5. A TextBox is a screen that allows the user to enter and edit text.
6. A TextField allows the user to enter and edit text. Multiple TextFields can be placed in a Form.
7. A DateField is an editable component for presenting date and time information. A DateField can be placed in a Form.
8. A Ticker acts as a scrollable display of text.

Figure 17.9 shows the user screens and its flow at run-time in the VMS application.

17.3 Implementing the Model

The two main components of the model for the VMS application were the communication manager and the media player. The communication manager implements the FrameworkSubscriber class of Peer2Me, which is necessary to use the framework. It handles all the communcation between the VMS application and the framework such as sending and recieving messages and search functions. The media player is the core of the VMS application. It is based on the media API of MIDP2. It allows the capture

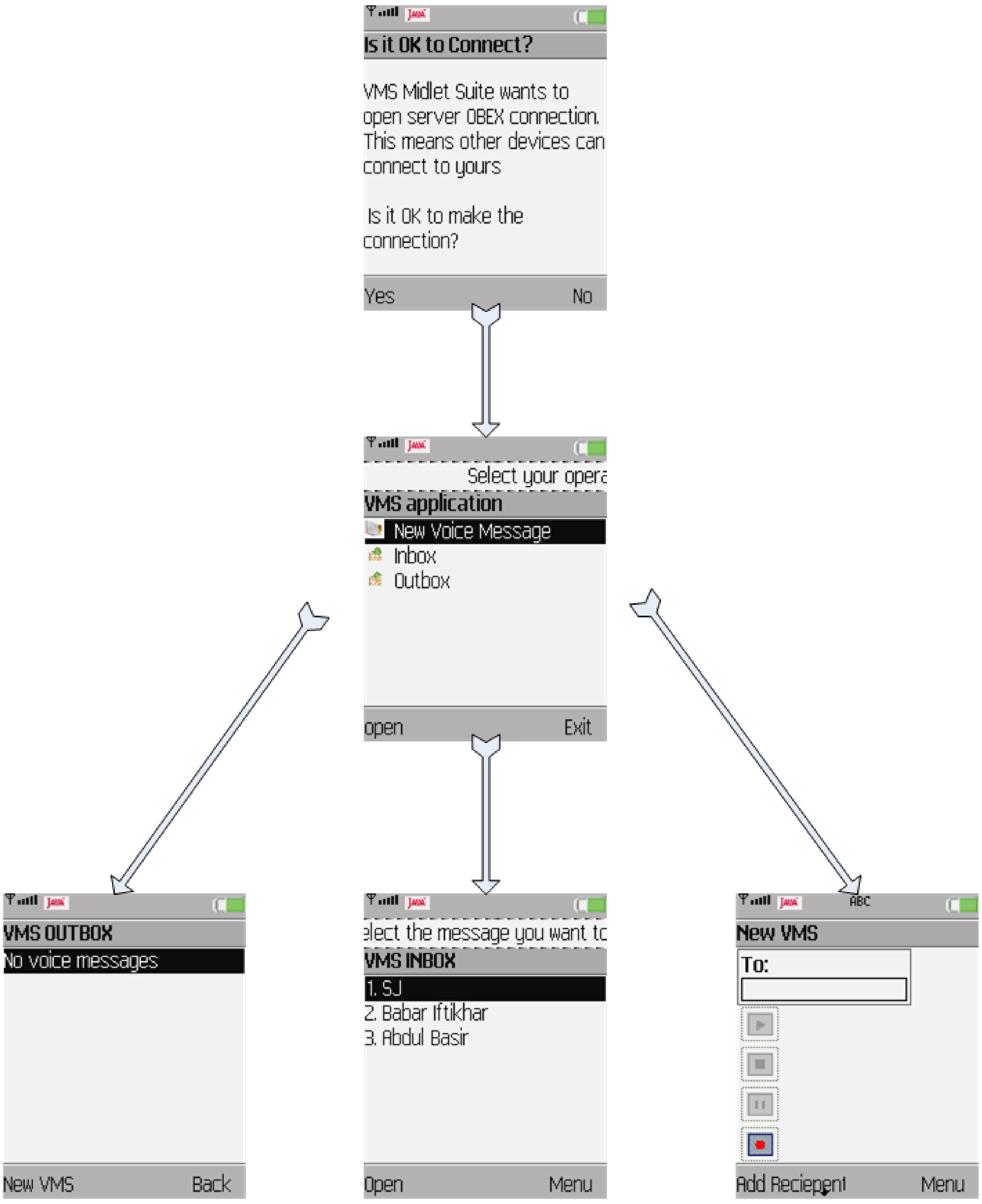


Figure 17.9: Screen shots of the VMS application.

of real time audio from the microphone of the phone. It is also provided with controls for playing, pausing and stopping the media, as shown in Figure 17.10.

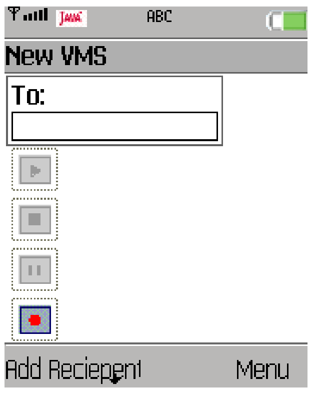


Figure 17.10: Screen shot of the media player.

17.4 Implementing the Controller

The VMSMidlet class functions as the main controller of the VMS application. It has a constructor, which initiallizes all of the components of the VMS application. It implements a CommandListener which handles the main events generated by the user of the application. It also provides methods for cleaning memory before the application exits, so that it is available for other applications on the phone.

17.5 Problems and Challenges

This section describes the problems and challenges faced during the implementation of the VMS application.

17.5.1 SDK Problems

It was hard to test some of the functionality of the application after its development due to some inherent limitations in the SDK's used. The SDK emulator for Sony Ericsson only have compilation support for audio capture and not run time support for it. So you need a real device to test this. Similarly on the sun emulator it is hard to emulate two phones comunicating via bluetooth.

17.5.2 Peer2Me Iterface Problems

PCM encoding was used for the audio format to capture voice. The data was stored as a byte array. For performance purposes, the format of this data was not changed to any other format. This posed a challenge since there was no external method/interface provided by the Peer2Me framework for such data. Although some internal methods do exist which support such formats and that were used to fulfill the requirements of the application.

Chapter 18

Testing

The VMS should be reliable and perform as expected. Adequate testing is important to ensure that these objectives are met. Different types of tests were performed to ensure the performance and workability of the application and increase user satisfaction of usability.

18.1 Black-box testing

Behavioural testing is sometimes also called ’functional’ or ’black-box’ testing because the software is treated as an opaque ’box’ that responds to external input without observing what it does internally. The success is determined by whether the software produces the results that are expected [45]. Most of the behavioral testing was done using the Sony Ericsson Java ME SDK [17] and Sun Java Wireless Toolkit [18]. This test was performed on the completion of each individual module e.g. the media player of the VMS. The media player was tested for various functionality such as recording of real time audio and its playback. It is worth noting here that the Sun Java Wireless toolkit does not support the simulation of two phones with different record stores. It was therefore not possible to test the sending and receiving of messages over bluetooth. The Sony Ericsson SDK on the other hand supported this bluetooth simulation but with another problem, it does not provide run-time support for capturing audio. For these reasons, the application had to be tested partly with both emulators. The audio capture functionality was tested and worked fine with the Sun Java Wireless Toolkit [18]. The message sending and receiving functionality was tested and worked fine on the Sony Ericsson Java ME SDK [17] by creating dummy messages with a similar format to that of a captured audio, as recording is not simulated in the SDK.

18.2 Structural Testing

Structural testing is also called 'white-box', 'clear-box', glass-box' and even 'no-box' testing. This test focuses on the internals of the software, where the functions are individually examined. This was an ongoing process during the implementation of the VMS application. Each piece of code was thoroughly tested for expected functionality throughout the development process.

18.3 System test / Usage test

After the completion of the implementation, the VMS midlet was tested on real phones. Two Sony Ericsson K750i phones were provided for this purpose. The application installed correctly on the phones and the peer searching functionality worked but it was not possible to record audio because the Sony Ericsson K750i is a Java Plateform 5 phone which does not support audio capture [36]. In order to test the midlet on a real phone, it requires a Java Plateform 6 or 7 phone such as Sony Ericsson W850i.

On request, one W850i was obtained from another student project for a few days. The W850i supports audio capture using J2ME, but unfortunately the code for capturing audio, `Manager.createPlayer("capture://audio?encoding=amr")` executed correctly on the phone, but did not capture any audio from the mic. This was known by checking the size of the OutputStream used to record the audio, which was always 0. Other coding techniques (`Manager.createPlayer("capture://devmic0?encoding=amr")`) were also used but the results were not the same as those on the emulator. Due to the lack of resources (time and equipment), the evaluation of the system is based on the emulation of the VMS on the emulator.

Chapter 19

Results and Evaluation

This chapter gives an evaluation of the reseach goals set for the project and results/goals achieved.

19.1 Answers to research questions

The VMS application started as a depth study in the Master Project done by Hassan [42]. The research questions/objectives initially set for the project can be found in Chapter 7. From the results obtained after the completion of the project, the following answers are given to the research questions.

RQ1 - How easy and convenient the Peer2Me framework is to utilize for the application developer?

Ans - The Peer2Me framework is very easy to use and performs its functionality as described. It has a very clear interface and is very well documented. It does not take much time to understand and start programming using the framework. It supports a wide variety of messaging formats which could be used to develop diverse applications.

RQ2 - To explore the feasibility of the Peer2Me framework for the development of voice/mp3 messaging/file transfers applications like VMS.

Ans - The Peer2Me framework is very well suited for the development of voice/mp3 messaging/file transfers applications like the VMS, though it was never previously used for such a system.

RQ3 - Discover technical limitations in the Peer2Me framework.

Ans - As also described in Section 17.5, the Peer2Me framework does not provide external interfaces to allow the attachment of byte arrays to a message. Internal

interfaces are available.
Currently the framework only supports bluetooth as the network medium.

RQ4 - Identify possible future extensions to the Peer2Me framework.

Ans - The Peer2Me framework can be extended with external interfaces to support the attachment of byte arrays to a message. This will allow an efficient way to send recorded messages.
The framework should be extended to support WLAN and GPRS to cover the most widely used mobile networks and allow the development of diverse and innovative applications.

RQ5 - How the VMS can be used to support mobile collaboration?

Ans - the VMS can be used in a number of situations to support collaboration among mobile workers/students. One such situation is in supporting distributed and mobile groups of distance learning students in the process of group formation, collaboration, coordination and communication [32]. Using the VMS system, these students can collaborate on the presentation artifacts, schedule meetings and other situations.

Mobile employees constitute one of the main resources in many industries. The work processes of these employees across different industries can be made more efficient by mobilizing existing enterprise applications and data. The VMS is a prototype system which can be developed according to the needs of these mobile workers for improving adhoc collaboration and data sharing.

19.2 Goals of the VMS application

During the requirements analysis of the VMS application, certain requirements were identified which can be found in Section 14.1. This section will analyse each of those requirements to evaluate whether they have been ediquetly achieved. It is important to note that this evaluation is based on the emulation of the VMS system on the emulator, since there were some problems (see Chapter 18) in testing the system on real phones.

FR1 - The VMS should be able to send and receive voice messages to other peers/nodes.

Achieved - This functionality has been tested on the emulators.

FR2 - The VMS should provide an interface for recording voice messages.

Achieved - A user friendly interface is available in the VMS application for recording as shown in Figure 19.1.

FR3 - The VMS should provide an interface to playback received messages.

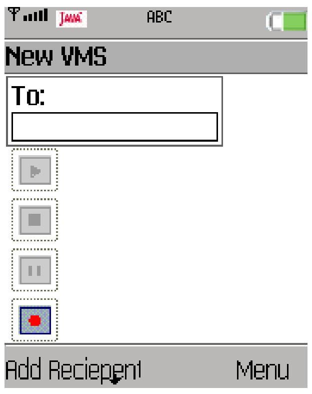


Figure 19.1: UI for recording.

Achieved - A user friendly interface is available in the VMS application for playback as shown in Figure 19.2.

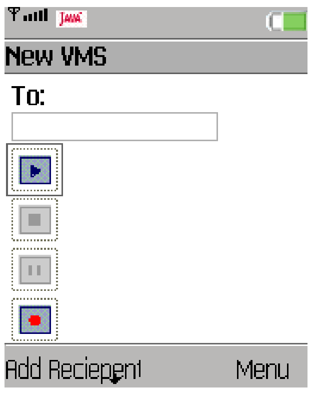


Figure 19.2: UI for playback.

FR4 - The VMS should use the Peer2Me framework.

Achieved - The VMS application uses the network and messaging services provided by the Peer2Me framework. The communication manager handles the communication between the VMS application and the Peer2Me framework.

FR5 - The VMS should store incomming voice messages.

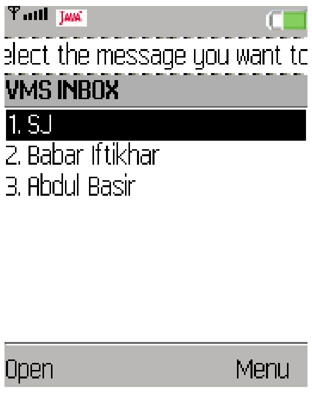
Achieved - All incomming voice messages are saved and displayed in the inbox view as shown in Figure 19.3.

FR6 - The VMS should alert the user of the arrival of a new message.

Achieved - The VMS application alerts the user with a message received screen on the arrival of a new voice message, as shown in Figure 19.4.

FR7 - The VMS should provide functionality to delete messages.

Achieved - Messages could be deleted both from the outbox and inbox as shown in Figure 19.5.

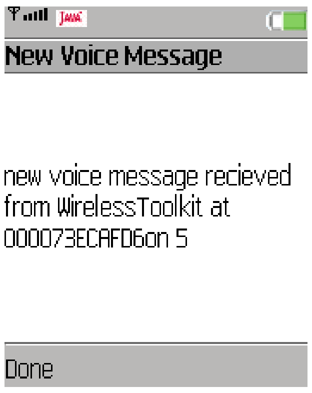


elect the message you want tc

VMS INBOX

1. SJ
2. Babar Iftikhar
3. Abdul Basir

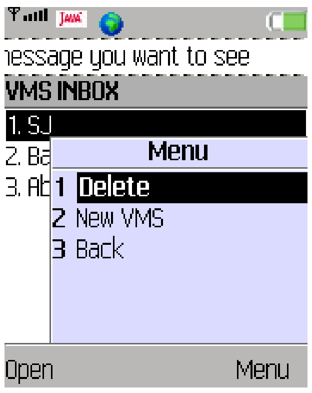
Figure 19.3: Inbox of the VMS.



New Voice Message

new voice message recieved from WirelessToolkit at 000073ECAFD6on 5

Figure 19.4: Message alert of the VMS.



ıessage you want to see

VMS INBOX

1. SJ
2. Ba
3. Ab

Figure 19.5: Deleting a message in the VMS.

FR8 - The VMS should provide functionality to forward a message to another node.

Achieved - Messages could be forwarded to other nodes by opening it from the inbox and selecting forward from the menu, as shown in Figure 19.6.

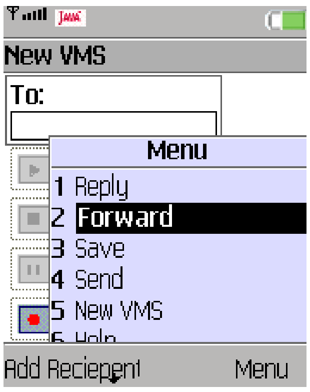


Figure 19.6: Forwarding a message to another node.

FR9 - The VMS should provide an inbox, where it is possible to view all received messages.

Achieved - All incomming voice messages are saved and displayed in the inbox view as shown in Figure 19.3.

FR10 - The VMS should provide an outbox, where it is possible to view all sent messages.

Achieved - All out-going voice messages are saved and displayed in the outbox view.

Part V

Conclusion and Further work

Chapter 20

Conclusion

The notion of ubiquitous computing is now being realized in every day life. The driving force behind this realization is the tremendous leap in technological development and the increased use of mobile devices in our daily life. In this project, we have tried to give the description and design of the Voice Messaging System (VMS), a step forward in the popular Short messaging service (SMS). The VMS is an attempt for a new era of communication which is intuitive to use and that adds an emotional and personal dimension to messaging on mobile phones.

The VMS is an application that allows end users to send voice messages to each other. It is based on a peer-to-peer technology and uses Bluetooth piconet (Section 9.4) and hence does not require the deployment of any infrastructure. It can be used on bluetooth enabled mobile phones and is free to use.

The design of the VMS is tightly coupled with the Peer2Me framework (Chapter 11). The Peer2Me framework provides communication facilities over bluetooth, which were exploited for the VMS. The message format of the VMS is extended from the one provided in the Peer2Me framework. More about the software architecture and design of the VMS can be read in Chapter 15 and 16.

The VMS can be used in a number of situations. It can be used by friends, family members or colleagues to form ad-hoc messaging networks, promoting informal ad-hoc collaboration. Recording a spoken message is less trouble and faster than typing a text SMS, and moreover, the sender of the message is free from language restrictions of the mobile phone keypad. The VMS is also meant to provide support to people with disabilities, having difficulty with typing text on mobile keypads. Chapter 21 describes possible future extensions to the VMS application.

Chapter 21

Further Work

This section describes further work that could be carried on with the VMS project in future.

21.1 Enhanced VMS

The VMS system has been developed as a prototype system for practically visualizing our idea. It has room for further work in design, scope and functionality. The VMS system can be extended with modules for video and still picture capture along with text messaging. This could prove to be a killer application for peer-to-peer multimedia messaging. With the development of new network modules for the Peer2Me framework [40], the VMS could be extended to use a variety of networks.

21.2 Inter-departmental collaboration

The current version of the Voice Messaging System (VMS) is targeted towards individual users. The system can be extended to a larger setting, such as within an enterprise or a University. In order to improve the network reliability and availability, it may require the installation of Bluetooth Transceivers/access points to obtain the desired capacity and coverage with in the physical surroundings where the service is required. It can be used for cheap inter-departmental collaboration.

The coverage can be further increased to geographically distributed areas by coupling the bluetooth Piconets with existing in-building Internet-enabled networks. Such a scheme is described by Yun Wu and Terence D. Todd in [48]. Another possiblilty is to provide the Peer2Me framework with capabilities to communicate over WLAN and GPRS. This will greatly increase the possibilities fot innovative application development.

21.3 Location awareness

The functionality of the VMS can be further extended to provide location sensitive information. In the inter-departmental setting described in Section 21.2, the address of the bluetooth transceivers can be used to locate colleagues. For instance peers could know if another peer is in his office, lab, class room, meeting room, gallery or by the coffee place. It will serve to increase awareness of colleagues for collaboration.

Part VI

Bibliography

Bibliography

[1] Bubble message, sms that speaks. retrieved october 24, 2006, from. http://telenor.com.pk/services/bubble_Message.php.

[2] Eclipse - an open development platform. retrieved september 1, 2006 from. http://www.eclipse.org.

[3] How stuff works. retrieved september 11, 2006 from. http://electronics.howstuffworks.com/bluetooth.htm/printable.

[4] Hp opencall voice short message service (sms) solution. retrieved october 24, 2006, from. http://h20208.www2.hp.com/opencall/library/solutions/ocmp/sms_sb.pdf.

[5] Java me technologies at a glance. retrieved november 4th, 2006, from. http://java.sun.com/javame/technologies/index.jsp.

[6] Jxta, webopedia, the #1 online encyclopedia dedicated to computer technology. retrieved october 17, 2006 from. http://www.webopedia.com/TERM/J/JXTA.html.

[7] Miktex....typesetting beautiful documents....retrieved september 2, 2006 from. http://www.miktex.org/.

[8] Mobile phone, from wikipedia, the free encyclopedia. retrieved may 03, 2007 from. http://en.wikipedia.org/wiki/Mobile_phone.

[9] Now you can talk, send and listen. retrieved october 25, 2006, from. http://www.digi.com.my/data_services/messaging/datamsg_bt_overview.do.

[10] The official bluetooth website. retrieved november 2nd, 2006, from. http://www.bluetooth.com/.

[11] Open source technology group. eclipseme. retrieved september 2, 2006 from. http://sourceforge.net/projects/eclipseme/.

[12] Open source technology group. texlipse. retrieved september 2, 2006 from. http://sourceforge.net/projects/texlipse/.

[13] peer-to-peer architecture, webopedia, the #1 onliine encyclopedia dedicated to computer technology. retrieved october 16, 2006 from. http://www.webopedia.com/TERM/p/peer_to_peer_architecture.html.

[14] Peer-to-peer, from wikipedia, the free encyclopedia. retrieved october 16, 2006 from. http://en.wikipedia.org/wiki/Peer-to-peer.

[15] S. microsystems. jsr 82: Javatmapis for bluetooth. retrieved october 10, 2006 from. http://www.jcp.org/en/jsr/detail?id=82.

[16] scatternet, webopedia, the #1 online encyclopedia dedicated to computer technology. retrieved october 17, 2006 from. http://www.webopedia.com/TERM/s/scatternet.html.

[17] Sony ericsson java me sdk . retrieved january 1, 2007 from. http://developer.sonyericsson.com/site/global/docstools/java/p_java.jsp.

[18] Sun microsystems. sun java wireless toolkit. retrieved september 1, 2006 from. http://java.sun.com/products/sjwtoolkit/.

[19] Ubiquitous computing. retrieved november 2nd, 2006, from. http://www.ubiq.com/hypertext/weiser/UbiHome.html.

[20] Voicemail. from wikipedia, the free encyclopedia. retrieved november 6th, 2006, from. http://en.wikipedia.org/wiki/Voicemail.

[21] The waterfall model. adrian als & charles greenidge, 2003. retrieved november 9th, 2006, from. http://scitec.uwichill.edu.bb/cmp/online/cs22l/waterfall_model.htm.

[22] waterfall model. retrieved november 9th, 2006, from. http://searchvb.techtarget.com/sDefinition/0,,sid8_gci519580,00.html.

[23] Wikipedia. mobile ad-hoc network. retrieved october 22, 2006, from. http://en.wikipedia.org/wiki/Mobile_ad-hoc_network.

[24] Wikipedia. personal area network. retrieved october 22, 2006, from. http://en.wikipedia.org/wiki/Personal_area_network.

[25] Annalee Newitz (July 2001). Sharing the data. metro, silicon valley's weekly newspaper. metro publishing inc. retrieved may 23, 2007 from. http://www.metroactive.com/papers/metro/07.12.01/work-0128.html.

[26] Zoltan Balazs. A practical use of the mvc pattern. retrieved may 24, 2007 from. http://www.codeproject.com/useritems/PraticalMvc.asp.

[27] Victor R. Basili. Experimental software engineering issues: Critical assessment and future directions. Technical report, pages 312, Proc. Int'l Workshop, Dagstuhl Castle, Germany,, September 14-18 1992. Springer Verlag LNCS 706.

[28] e Zest Technologies Pvt. Ltd. J2me (javaŹ 2 platform, micro edition) application development. retrieved may 24, 2007 from. `www.e-zest.net/j2me.html`.

[29] George H. Forman and John Zahorjan. The challenges of mobile computing. technical report. Technical report, University of Washington, 1994.

[30] Dustin Preuitt Thaddeus G. C. Thompson Stephen Fickas Gerd Kortuem, Jay Schneider and Zary Segall. When peer-to-peer comes face-to-face: Collaborative peer-to-peer computing in mobile ad hoc networks. technical report. Technical report, Department of Computer and Information Science, University of Oregon, 2001.

[31] Eric Giguere. Programming strategies for small devices, chapter 3. Technical report, Sun developer network, February 2001.

[32] Mayende Godfrey. Use of mobile technology to support collaboration of students in developing countries. Technical report, IDI, NTNU, 2007.

[33] C. Gutwin and S. Greenberg. A descriptive framework of workspace awareness for real-time groupware. Technical report, Computer Supported Cooperative Work, Kluwer Academic Press, 2002.

[34] Mike Rowehl: This is Mobility. Partial overview of mobile content tools. retrieved may 24, 2007 from. `http://www.thisismobility.com/blog/?p=199`.

[35] Carl-Henrik Wolf Lund and Michael Sars Norum. The peer2me framework, a framework for mobile collaboration on mobile phones. master's thesis. Technical report, NTNU, 2005.

[36] Mobile-utopia. Sony ericsson system properties. retrieved may 22, 2007 from. `http://www.mobile-utopia.com/calibrator/SonyEricssonK750i%7Bfs%7DR1DB%7Bfs%7DSN359302007282673+Java%7Bfs%7DSEMC-Java%7Bfs%7D2.0+Profile%7Bfs%7DMIDP-2.0+Configuration%7Bfs%7DCLDC-1.1+UNTRUSTED%7Bfs%7D1.0---SonyEricssonK750i%7Bfs%7DR1DB001`.

[37] R.S.Fish R. R.E.Kraut and B.L.Chalfonte. Informal communication in organizations: form, functions and technology. people's reactions to technology in factories, offices and aerospace. Technical report, 1999.

[38] R.E.Johnson and B.Foote. Designing reusable classes. journal of object-oriented programming. Technical report, 1988.

[39] Kim Petter Saxlund and Tommy Bjørnsgård. Evaluation of peer2me, depth study 2005. Technical report, NTNU, 2005.

[40] Kim Petter Saxlund and Tommy Bjørnsgård. The improved peer2me framework, a flexible framework for mobile collaboration. Technical report, NTNU, 2006.

[41] Sun Developer Network (SDN). Mobile media api (mmapi); jsr 135 overview. retrieved may 24, 2007 from. `http://java.sun.com/products/mmapi/overview.html`.

[42] Hassan Syed Shah. Voice messaging system vms using the peer2me framework. Technical report, IDI NTNU, 2006.

[43] Hassan Syed Shah. Implementation of the voice messaging system vms using the peer2me framework. Technical report, IDI NTNU, 2007.

[44] Arne Sølvberg. Trondheim wireless broadband commons. retrieved april 11, 2007 from. `http://video.google.com/videoplay?docid=-9041070853343993488&q=wireless+trondheim`.

[45] Richard Smith. Testing short message service applications. Technical report, Business briefing: Wireless Technology, 2003.

[46] Yashraj Chauhan. Vikas Gupta, Avnish Dass. Cracking the code. wireless programming with j2me. Technical report, Dreamtech Software Team, 2002.

[47] Wikipedia. Bittorrent. retrieved may 23, 2007 from. `http://en.wikipedia.org/wiki/BitTorrent`.

[48] Yun Wu and Terence D. Todd. Link sharing in high capacity bluetooth voice access networks. Technical report, McMaster University, Hamilton, Ontario, CANADA, 2004.

Part VII

Appendix

Appendix A

VMS Source Code

A.1 VMSMidlet.java

```
import java.io.ByteArrayOutputStream;
import java.io.DataOutputStream;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
import javax.microedition.rms.RecordStore;
import peer2me.message.Message;

public class VMSMidlet extends MIDlet implements CommandListener{
    public Display display;
    Alert deleteAlert;
    boolean inboxFlag=false;
    Displayable previousScreen;
//  ++++++++++++++++++++++++RMS Code++++++++++++++++++++++++++++++++++++++++++++
    public RecordStore inboxStore =null;
    public RecordStore outboxStore =null;
//  *******************create views used by the application**********************
public VmsView vms= null; // first screen, giving access to newvms, inbox or outbox
public InboxView inbox= null;
public OutboxView outbox= null;
public NewVmsView newVms= null;
public Communicator comm;
// ********************create commands used in the application*******************
public Command exitCommand = new Command("Exit",Command.EXIT,10);
public Command backCommand = new Command("Back",Command.BACK, 0);
private Command helpCommand = new Command("Help",Command.HELP, 60);
private Command outboxCommand = new Command("Outbox",Command.SCREEN, 60);
private Command inboxCommand = new Command("Inbox",Command.SCREEN, 60);
public Command newVmsCommand = new Command("New VMS",Command.SCREEN, 60);
private Command sendCommand = new Command("Send",Command.SCREEN, 2);
    private Command saveCommand = new Command("Save",Command.SCREEN, 2);
    //Item commands ---> related to List, inbox and outbox
    public Command openCommand = new Command("Open",Command.ITEM, 1);
    public Command deleteCommand = new Command("Delete",Command.ITEM, 60);
private Command addReciepentCommand = new Command("Add Reciepent",Command.ITEM,60);
private Command forwardCommand = new Command("Forward",Command.ITEM, 60);
private Command replyCommand = new Command("Reply",Command.ITEM, 60);
// Alert Commands
private Command okCommand = new Command("OK", Command.OK,60);
private Command cancelCommand = new Command("Cancel", Command.CANCEL,60);

public VMSMidlet() {
vms = new VmsView(this);
        comm = new Communicator(this);
newVms = new NewVmsView(comm);
inbox = new InboxView(inboxStore,this);
outbox = new OutboxView(outboxStore, this);
display = Display.getDisplay(this);
//      INBOX Commands................................................
inbox.addCommand(backCommand);
inbox.addCommand(newVmsCommand);
```

```
//      OUTBOX Commands...............................................
outbox.addCommand(newVmsCommand);
outbox.addCommand(backCommand);
//      NEWVMS Commands................................................
newVms.addCommand(backCommand);
newVms.addCommand(helpCommand);
newVms.addCommand(addReciepentCommand);
newVms.addCommand(saveCommand);
newVms.addCommand(exitCommand);
//      COMMAND LISTENER for ALL. except VMS VIEW........................
inbox.setCommandListener(this);
outbox.setCommandListener(this);
newVms.setCommandListener(this);
display.setCurrent(vms);
}
protected void startApp() throws MIDletStateChangeException {
// used for starting and restarting a midlet, and catching exceptions

}
protected void pauseApp() {
// used to release resources, by initializing to null

}
protected void destroyApp(boolean arg0)  {
// used to perform cleanup operations before exiting ie network connections
// GUI commponents and database records.
display.setCurrent((Displayable)null);
        vms = null;
inbox =  null;
outbox =  null;
        newVms =  null;
}
public void commandAction(Command c, Displayable s){
if (c== exitCommand){
comm.framework.clean(); // release resources.....
destroyApp(true);
notifyDestroyed();
}
if (c== newVmsCommand){
newVms.removeCommand(newVmsCommand);
setDisplay(0);
}
if (c== inboxCommand){
display.setCurrent(inbox);
}
if (c== outboxCommand){
display.setCurrent(outbox);
}
if (c== backCommand){
if(s==newVms)
display.setCurrent(previousScreen);
else
display.setCurrent(vms);
}
// INBOX COMMANDS*****************************************************************
if(c==openCommand & s== inbox){
setDisplay(0); // change display to player (newVMS view)
inbox.openMessage();
newVms.addCommand(replyCommand);
newVms.addCommand(forwardCommand);
newVms.addCommand(newVmsCommand);
}

if(c==deleteCommand & s== inbox){
deleteAlert = new Alert("Delete", "Are you sure you want to delete "+ inbox.getString(inbox.getSelectedIndex())+ " ?",null,AlertType.CONFIRMATION);
deleteAlert.addCommand(okCommand);
deleteAlert.addCommand(cancelCommand);
deleteAlert.setCommandListener(this);
inboxFlag=true;
previousScreen =display.getCurrent();
display.setCurrent(deleteAlert,display.getCurrent());
}

if(c==okCommand & s== deleteAlert){
if(inboxFlag==true){
inbox.deleteMessage();
display.setCurrent(inbox);
}else if(inboxFlag==false){
outbox.deleteMessage();
display.setCurrent(outbox);
}
}
if(c==cancelCommand & s== deleteAlert){
display.setCurrent(previousScreen);
}

// OUTBOX COMMANDS******************************************************************
if(c==openCommand & s== outbox){
setDisplay(0); // change display to player (newVMS view)
outbox.openMessage();
newVms.addCommand(forwardCommand);
newVms.addCommand(newVmsCommand);
}
if(c==deleteCommand & s== outbox){
deleteAlert = new Alert("Delete", "Are you sure you want to delete "+ outbox.getString(outbox.getSelectedIndex())+ " ?",null,AlertType.CONFIRMATION);
deleteAlert.addCommand(okCommand);
deleteAlert.addCommand(cancelCommand);
deleteAlert.setCommandListener(this);
inboxFlag=false;
previousScreen =display.getCurrent();
display.setCurrent(deleteAlert,display.getCurrent());
}

// NewVMS COMMANDS*********************************************************************
if(c==replyCommand & s==newVms){
newVms.reply();
newVms.removeCommand(replyCommand);
newVms.removeCommand(forwardCommand);
}
if(c==forwardCommand & s==newVms){
newVms.forward();
newVms.removeCommand(replyCommand);
newVms.removeCommand(forwardCommand);
}
if(c==addReciepentCommand){
comm.framework.search();
//////Display alert while searching.......
Alert searchAlert = new Alert("Searching peers..", "this operation might take a few seconds, please be patient!",null,AlertType.INFO);
searchAlert.setTimeout(Alert.FOREVER);
display.setCurrent(searchAlert);
newVms.addCommand(sendCommand);
}
if (c==sendCommand){

Message message = new Message();
Alert sentAlert = new Alert("Voice message", "sending message..",null,AlertType.CONFIRMATION);
Gauge indicator = new Gauge(null,false,Gauge.INDEFINITE,Gauge.CONTINUOUS_RUNNING);
sentAlert.setIndicator(indicator);
display.setCurrent(sentAlert);
try{
message.addObjectBytes("Mes", newVms.myPlayer.recordedSoundArray);
}catch(Exception err){
                //Display alert if an error occurs.......
     Alert errorAlert = new Alert("","Error adding recording to message " + err.getMessage(),null,AlertType.ERROR);
     errorAlert.setTimeout(Alert.FOREVER);

     display.setCurrent(errorAlert);
            }
message.addRecipient(comm.currentlySelectedNode);
comm.framework.sendMessage(message);
sentAlert.setString("Congrats: Message sent!");
// save the voice message in outbox after sending..
saveFile();
}

if(c==helpCommand){
Alert helpAlert = new Alert("VMS Help", "This is the online help of the VMS System. In order to send a message to a nearby peer, " +
"first record your voice message by going to New VMS screen. The next step is to add the recepient to whom you want to send " +
"the message. The send option is now visible in the menu and pressing it will send your recorded message to the selected recepient."
,null,AlertType.INFO);
helpAlert.setTimeout(Alert.FOREVER);//modal alert.....
// more to write about Help............
display.setCurrent(helpAlert);
}

}
//        method called from Vms view, to display the screen/ view selected from list
public  void setDisplay(int option){

if (option== 0){
```

```
if((display.getCurrent()!= newVms)){
previousScreen =display.getCurrent();
}
display.setCurrent(newVms);
newVms.initialize();
}
if (option== 1){
display.setCurrent(inbox);
}
if (option== 2){
display.setCurrent(outbox);
}
}
//      code for storing sent message in outbox
private void saveFile(){
//open/create record store****************************************
        try{
            outboxStore =RecordStore.openRecordStore("VMSoutbox",true);
        }catch(Exception err){
//                Display alert if an error occurs.......
Alert errorAlert = new Alert("","Error creating RMS (VMSourtbox) " + err.getMessage(),null,AlertType.ERROR);
errorAlert.setTimeout(Alert.FOREVER);
display.setCurrent(errorAlert);
        }
        //write data to record store****************************************
        try{
            // data to store............
            byte[] outputRecord = newVms.myPlayer.recordedSoundArray;
            String nick = comm.currentlySelectedNode.getNodename();
            ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
            DataOutputStream outputDataStream = new DataOutputStream(outputStream);
            outputDataStream.writeUTF(nick);
            outputDataStream.write(outputRecord);
            outputDataStream.flush();
            outputRecord= outputStream.toByteArray();
            outboxStore.addRecord(outputRecord,0,outputRecord.length);
            // release resources
            outputStream.close();
            outputDataStream.close();
        }catch(Exception err){
//          Display alert if an error occurs.......
Alert errorAlert = new Alert("","Error adding record to outbox " + err.getMessage(),null,AlertType.ERROR);
errorAlert.setTimeout(Alert.FOREVER);
display.setCurrent(errorAlert);
        }
//*****************close record store, release resources**************************
        try{
         outboxStore.closeRecordStore();
        }catch(Exception err){
//         Display alert if an error occurs.......
Alert errorAlert = new Alert("","Error closing outbox " + err.getMessage(),null,AlertType.ERROR);
errorAlert.setTimeout(Alert.FOREVER);
display.setCurrent(errorAlert);
        }
        // update outboxView
        outbox.update();
}
}
```

A.2 NewVmsView.java

```
import javax.microedition.lcdui.*
import peer2me.framework.Framework;
import peer2me.message.Message;

public class NewVmsView extends Form implements ItemCommandListener {
//Flags to change button pictures
boolean isPlay;
boolean isStop;
boolean isRecord;
boolean isPause;
Communicator comm=null;
PlayerRecorder myPlayer; // instance of player Recorder
public TextField recipient;
    private Gauge progress;

    Image recordImage;
Image playImage;
Image stopImage;
Image pauseImage;
Image recordDownImage;
Image playDownImage;
Image stopDownImage;
Image pauseDownImage;

// image item used as buttons for the player recorder
ImageItem recordBtn;
ImageItem playBtn;
ImageItem stopBtn;
ImageItem pauseBtn;

    public StringItem messageItem;
// player, recorder commands.......
private Command recordCommand = new Command("Record Message",Command.ITEM, 1);
private Command stopCommand = new Command("Stop",Command.ITEM, 60);
private Command pauseCommand = new Command("Pause",Command.ITEM, 60);
private Command playCommand = new Command("Play",Command.ITEM, 60);

public NewVmsView(Communicator c) {
super("New VMS ");
this.comm = c;
recipient = new TextField("To:", null, 120, TextField.ANY);
        recipient.setPreferredSize(120, 35);
progress = new Gauge("progress bar", false, 200, 0);
        messageItem = new StringItem(" ", "Click record to start recording.");
        myPlayer= new PlayerRecorder(this);
        try{
recordImage = Image.createImage("/images/recordBtn.png");
playImage = Image.createImage("/images/playBtn.png");
stopImage = Image.createImage("/images/stopBtn.png");
pauseImage = Image.createImage("/images/pauseBtn.png");
recordDownImage = Image.createImage("/images/recordBtnd.png");
playDownImage = Image.createImage("/images/playBtnd.png");
stopDownImage = Image.createImage("/images/stopBtnd.png");
pauseDownImage = Image.createImage("/images/pauseBtnd.png");
} catch (java.io.IOException err){
//          Display alert if an error occurs.......
Alert errorAlert = new Alert("","Error: reading images (NewVmsView) " + err.getMessage(),null,AlertType.ERROR);
errorAlert.setTimeout(Alert.FOREVER);
comm.midlet.display.setCurrent(errorAlert);
        }
// instantiate butons
recordBtn = new ImageItem(null,recordImage,ImageItem.LAYOUT_LEFT,"Record");
playBtn = new ImageItem(null,playDownImage,ImageItem.LAYOUT_LEFT|ImageItem.LAYOUT_NEWLINE_AFTER,"Play");
stopBtn = new ImageItem(null,stopDownImage,ImageItem.LAYOUT_LEFT|ImageItem.LAYOUT_NEWLINE_AFTER,"Stop");
pauseBtn = new ImageItem(null,pauseDownImage,ImageItem.LAYOUT_LEFT|ImageItem.LAYOUT_NEWLINE_AFTER,"Pause");

// set default commands for buttons, invoked by select key
recordBtn.setDefaultCommand(recordCommand);// default command for the record button
playBtn.setDefaultCommand(playCommand);// default command for the play button
stopBtn.setDefaultCommand(stopCommand);// default command for the stop button
pauseBtn.setDefaultCommand(pauseCommand);// default command for the pause button

//set item command listener for the buttons
recordBtn.setItemCommandListener(this);
playBtn.setItemCommandListener(this);
stopBtn.setItemCommandListener(this);
pauseBtn.setItemCommandListener(this);
recipient.setItemCommandListener(this);

// attach buttons to the newVms form
        this.append(recipient);
this.append(playBtn);
this.append(stopBtn);
this.append(pauseBtn);/**/
this.append(recordBtn);
        this.append(messageItem);
        this.append(progress);

initializeButtonStates();//set initial states for player buttons
}
// Item Command listener for player, recorder buttons...........
public void commandAction(Command c, Item item){
if (item == recordBtn & (recordBtn.getImage()==recordImage)){
if(!isRecord){
isRecord =true;
isPlay = false;
isStop=false;
isPause=false;
recordBtn.setImage(recordDownImage);
stopBtn.setImage(stopImage);
pauseBtn.setImage(pauseImage);
playBtn.setImage(playDownImage);
                messageItem.setText("recording...");
                myPlayer.record();
}
}
if (item == playBtn){
if(!isPlay & (playBtn.getImage()==playImage)){
isPlay=true;
isRecord=false;// ambiguos......check if works correctly
isPause=false;
isStop=false;
playBtn.setImage(playDownImage);
recordBtn.setImage(recordDownImage);
pauseBtn.setImage(pauseImage);
stopBtn.setImage(stopImage);
    messageItem.setText("playing...");
                myPlayer.play();
}
}
if (item == stopBtn){
if(!isStop){
isStop=true;
messageItem.setText("stopped...");
if(isRecord & !isPause){
                stopped();
myPlayer.stop(2);
}
else if(isPlay){
                    stopped();
// System.out.println("Stop button pressed after play");
                    myPlayer.stop(3);
}
else if(isPause & (!isRecord)){
                    stopped();
// System.out.println("Stop button pressed after pause on record ");
                    myPlayer.stop(2);
}
else if(isPause & (!isPlay)){
                    stopped();
// System.out.println("Stop button pressed after pause on play ");
myPlayer.stop(3);
}
}
}
if (item == pauseBtn){
if(!isPause){
isPause=true;
                messageItem.setText("paused...");
if(isRecord){
pauseBtn.setImage(pauseDownImage);
recordBtn.setImage(recordImage);
stopBtn.setImage(stopImage);
isStop=false;
isRecord=false;
myPlayer.pause(4);
                }
if(isPlay){
isPlay= false;
isRecord=true;
isStop=false;
pauseBtn.setImage(pauseDownImage);
playBtn.setImage(playImage);
recordBtn.setImage(recordDownImage);
stopBtn.setImage(stopImage);
myPlayer.pause(5);
}
}
}
}

//initialize player button states at startup and after sending a message
public void initializeButtonStates(){
//buttons in false state are enabled.....
isPlay=true; //
isRecord=false;
```

```
isStop=true;
isPause= true;
recordBtn.setImage(recordImage);
stopBtn.setImage(stopDownImage);
pauseBtn.setImage(pauseDownImage);// nothing to play initially
playBtn.setImage(playDownImage);
}
    public void stopped(){
        // same code placed in a routine, is called from different places and also from recorder at the end of record.
            messageItem.setText("stopped...");
            isPlay=false;
            isRecord=false;
            isPause=true;
            pauseBtn.setImage(pauseDownImage);
            playBtn.setImage(playImage);
            stopBtn.setImage(stopDownImage);
            recordBtn.setImage(recordImage);
    }

    public void playMessage(String nick,byte[] message,String source){
        initializePlayButtons();
     recipient.setString(nick);
     if(source=="inbox")
     recipient.setLabel("From:");
     else if(source=="outbox")
     recipient.setLabel("To:");
     myPlayer.recordedSoundArray= message;
    }
    // method called when ever a message is opened from inbox/outbox, to be played.
    // initially has only play button enabled......
private void initializePlayButtons() {
// buttons in false state are enabled.....
isPlay=false; //
isRecord=true;
isStop=true;
isPause= true;
recordBtn.setImage(recordDownImage);
stopBtn.setImage(stopDownImage);
pauseBtn.setImage(pauseDownImage);// nothing to play initially
playBtn.setImage(playImage);
}
// called from VMSmidlet, to reset the recipient field for sending a new VMS
public void initialize() {
recipient.setString("");
recipient.setLabel("To:");
initializeButtonStates();
}
public void reply() {
recipient.setLabel("To:");
this.initializeButtonStates();
}
public void forward() {
recipient.setString("");
recipient.setLabel("To:");
}
}
```

A.3 InboxView.java

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;

import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.List;
import javax.microedition.lcdui.Ticker;
import javax.microedition.rms.RecordEnumeration;
import javax.microedition.rms.RecordStore;
import javax.microedition.rms.RecordStoreNotFoundException;

import peer2me.message.Message;
```

```
public class InboxView extends List{

RecordStore inboxStore = null;
    RecordEnumeration re=null;
    VMSMidlet mymid;

public InboxView(RecordStore ibSt, VMSMidlet mid) {
super("VMS INBOX",List.IMPLICIT);
// show instructions in a ticker oject
Ticker inboxTicker = new Ticker("Select the message you want to see");
this.setTicker(inboxTicker);
this.setFitPolicy(TEXT_WRAP_OFF);// wrap message headers on new lines, show full header
this.inboxStore = ibSt;
        this.mymid = mid;
        this.update();
}
public void update() {
 // read nicks from VMSinbox, and display them in a list;
        try{
         inboxStore =RecordStore.openRecordStore("VMSinbox",true);
         if(inboxStore.getNumRecords()!=0){
         byte[] byteInputData;
         ByteArrayInputStream inputStream = null;
         DataInputStream inputDataStream;

         re = inboxStore.enumerateRecords(null,null,false);
         this.deleteAll();
         while(re.hasPreviousElement()){
         int temp = re.previousRecordId();//.nextRecordId();

                    byteInputData= new byte[inboxStore.getRecordSize(temp)];
                    inputStream = new ByteArrayInputStream(byteInputData);
                    inputDataStream = new DataInputStream(inputStream);

                    inboxStore.getRecord(temp , byteInputData,0);

                    this.append(temp + " " + inputDataStream.readUTF(),null);
         }
           inputStream.close();
         }
            inboxStore.closeRecordStore();
        }
        catch (RecordStoreNotFoundException err) {
//          Display alert if an error occurs.......
Alert errorAlert = new Alert("","Error: recordstore inbox not found (InboxView) " + err.getMessage(),null,AlertType.ERROR);
errorAlert.setTimeout(Alert.FOREVER);
mymid.display.setCurrent(errorAlert);

        }catch(Exception err){
//          Display alert if an error occurs.......
Alert errorAlert = new Alert("","Error: reading VMSinbox (InboxView) " + err.getMessage(),null,AlertType.ERROR);
errorAlert.setTimeout(Alert.FOREVER);
mymid.display.setCurrent(errorAlert);
        }
        if(this.size()==0){
         this.append("No voice messages",null);
         this.removeCommand(mymid.openCommand);
this.removeCommand(mymid.deleteCommand);}
        else{
            this.addCommand(mymid.openCommand);
     this.addCommand(mymid.deleteCommand);
         this.setSelectCommand(mymid.openCommand);
        }
}
public void openMessage() {
//  get the selected index, and extract that message from the RMS and pass it to newVmsView
// int i = this.getSelectedIndex()+1;// adding 1, to start records from 1 and not 0

int i = (this.getString(this.getSelectedIndex())).charAt(0)-48;
String FromNick;
try{
inboxStore =RecordStore.openRecordStore("VMSinbox",false);

byte[] byteInputData= new byte[inboxStore.getRecordSize(i)];
            ByteArrayInputStream inputStream = new ByteArrayInputStream(byteInputData);
            DataInputStream inputDataStream= new DataInputStream(inputStream);
            inboxStore.getRecord(i , byteInputData,0);
            FromNick = inputDataStream.readUTF();
            byte[] message = new byte[inboxStore.getRecordSize(i)];
            inputDataStream.read(message);
            inputStream.close();
            inboxStore.closeRecordStore();
            mymid.newVms.playMessage(FromNick,message,"inbox");
        }
        catch(Exception err){
//          Display alert if an error occurs.......
Alert errorAlert = new Alert("","Error: in openMessage method (InboxView) " + err.getMessage(),null,AlertType.ERROR);
errorAlert.setTimeout(Alert.FOREVER);
mymid.display.setCurrent(errorAlert);
        }
}
public void deleteMessage() {
// int i = this.getSelectedIndex()+1;// adding 1, to start records from 1 and not 0
int i = (this.getString(this.getSelectedIndex())).charAt(0)-48;
try{
inboxStore =RecordStore.openRecordStore("VMSinbox",false);
            inboxStore.deleteRecord(i);
            inboxStore.closeRecordStore();
            this.update();
        }
        catch(Exception err){
//          Display alert if an error occurs.......
Alert errorAlert = new Alert("","Error: deleteMessage method (InboxView) " + err.getMessage(),null,AlertType.ERROR);
errorAlert.setTimeout(Alert.FOREVER);
mymid.display.setCurrent(errorAlert);
        }
}
}
```

A.4 PlayerRecorder.java

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.media.*;
import javax.microedition.media.control.RecordControl;
import javax.microedition.midlet.*;

public class PlayerRecorder implements PlayerListener{
    public Player recorder=null;
    public Player player=null;
    public byte[] recordedSoundArray = null;
    NewVmsView myView; // reference to view to change UI components
    RecordControl rc;
    ByteArrayOutputStream output;
    public boolean recPaused;
    public boolean recordingStopped; // flag to check stop on recordto avoid calling it twice
    public int MAX_LIMIT= 170000; // max limit for recording
    private PlayerListener myListener;

    public PlayerRecorder(NewVmsView nvms) {
        myView = nvms;
        recPaused =false;
        myListener = this;
    }

    public void record(){
        new Thread(new Runnable(){
            public void run(){
                if(player!= null) player.close(); // close the previous player object, if any, when recording something new
                // resuming from a pause on recording
                if(recPaused){
                    rc.startRecord();
                    recPaused=false;
                }
                else {
                    try {
                    recordedSoundArray = null; // reset
                    recordingStopped=false; // reset flag on new recording
                    //recorder = Manager.createPlayer("capture://audio?encoding=pcm"); // create a recorder to capture audio from mic
                    recorder = Manager.createPlayer("capture://audio?encoding=amr");
                    recorder.addPlayerListener(myListener);
                    recorder.realize();
                    // get the RecordControl over this Player
                    rc = (RecordControl)recorder.getControl("RecordControl");
                    // create an OutputStream which the RecordControl will use
                    // to write write the recorded data.
                    output = new ByteArrayOutputStream();
                    rc.setRecordSizeLimit(MAX_LIMIT); // recording on a maximum limit of bytes, rather than sleeping the thread
                    rc.setRecordStream(output);
                    // start the recording
                    rc.startRecord();
                    recorder.start();

                    } catch (IOException ioe) {
//                      Display alert if an error occurs.......
             Alert errorAlert = new Alert("","Error: record IOException in record method (PlayerRecorder) " +
             ioe.getMessage(),null,AlertType.ERROR);
             errorAlert.setTimeout(Alert.FOREVER);
             myView.comm.midlet.display.setCurrent(errorAlert);

                    } catch (MediaException me) {
//                      Display alert if an error occurs.......
             Alert errorAlert = new Alert("","Error: record MediaException in record method (PlayerRecorder) " +
             me.getMessage(),null,AlertType.ERROR);
             errorAlert.setTimeout(Alert.FOREVER);
             myView.comm.midlet.display.setCurrent(errorAlert);
                    } catch(OutOfMemoryError om){
//                      Display alert if an error occurs.......
                Alert errorAlert = new Alert("","Error: Out of memory exception in record method (PlayerRecorder) " +
                om.getMessage(),null,AlertType.ERROR);
                errorAlert.setTimeout(Alert.FOREVER);
                myView.comm.midlet.display.setCurrent(errorAlert);
                    }
                }
            }
        }).start();
    }

    public void play(){
        if (player == null || player.getState()==player.CLOSED){
            try {
                ByteArrayInputStream recordedInputStream = new ByteArrayInputStream(recordedSoundArray);
                player= Manager.createPlayer(recordedInputStream,"audio/x-wav");// (recordedInputStream,"audio/basic");
                player.addPlayerListener(this);
                player.prefetch();
                player.start();
            }  catch (IOException ioe) {
                System.out.println("IO exception in play.... " + ioe.getMessage());
//              Display alert if an error occurs.......
        Alert errorAlert = new Alert("","Error: IO exception in play method (PlayerRecorder) " + ioe.getMessage(),null,AlertType.ERROR);
        errorAlert.setTimeout(Alert.FOREVER);
        myView.comm.midlet.display.setCurrent(errorAlert);
            } catch (MediaException me) {
//              Display alert if an error occurs.......
        Alert errorAlert = new Alert("","Error: Media exception in play method (PlayerRecorder) " + me.getMessage(),null,AlertType.ERROR);
        errorAlert.setTimeout(Alert.FOREVER);
        myView.comm.midlet.display.setCurrent(errorAlert);
            }
        }
        else {
            try{
                // player was paused and is played again
                //start the player from current media time
                player.start();
            }catch(MediaException me){
//              Display alert if an error occurs.......
        Alert errorAlert = new Alert("","Error: Media exception in play method (PlayerRecorder) " + me.getMessage(),null,AlertType.ERROR);
        errorAlert.setTimeout(Alert.FOREVER);
        myView.comm.midlet.display.setCurrent(errorAlert);
            }
        }
     }

    public void stop(int m){
            if(m==2){
//          stop called on record....
            recordingStopped = true;
            try {
                rc.commit();
                // save the recordedData in a byte array
                recordedSoundArray = output.toByteArray();
                output.close();
                // close the player
                recorder.close();
            } catch (OutOfMemoryError ome){
//              Display alert if an error occurs.......
        Alert errorAlert = new Alert("","Error: out of memory in stop method (PlayerRecorder) " + ome.getMessage(),null,AlertType.ERROR);
        errorAlert.setTimeout(Alert.FOREVER);
        myView.comm.midlet.display.setCurrent(errorAlert);

            } catch (IOException ex) {
//              Display alert if an error occurs.......
        Alert errorAlert = new Alert("","Error: IO exception in stop method (PlayerRecorder) " + ex.getMessage(),null,AlertType.ERROR);
        errorAlert.setTimeout(Alert.FOREVER);
        myView.comm.midlet.display.setCurrent(errorAlert);
        }
        }

        else if(m==3){
//          stop called on play
            if(player!=null && player.getState()==player.STARTED){
                    player.close();
            }
        }
    }

    public void pause(int m){
        if(m==4){
            // pause called on record....
            if(recorder!=null && recorder.getState()==recorder.STARTED){
//                  pause the recorder. Note that recorder does not rewind to beginning when stop is called
                    recPaused=true; // pause flag
                    rc.stopRecord();
           }
        }
        else if(m==5){
            // pause called on play
            if(player!=null && player.getState()==player.STARTED){
                try {
//                  stop the player. Note that player does not rewind to beginning when pause is called
                    player.stop();
                }catch(MediaException me){
//                  Display alert if an error occurs.......
            Alert errorAlert = new Alert("","Error: Media exception in pause method (PlayerRecorder) " + me.getMessage(),null,AlertType.ERROR);
            errorAlert.setTimeout(Alert.FOREVER);
            myView.comm.midlet.display.setCurrent(errorAlert);
                }
            }
        }
    }

    public void playerUpdate(Player player, String event, Object eventData) {
            // player listener, fires events on completion of media in play mode, as well as record mode.
            if(player==player && event==END_OF_MEDIA){
                player.close();
                myView.stopped(); // reset buttons in UI after stopped playing file
            }
            if(player==recorder && event==RECORD_STOPPED && recordingStopped == false && recPaused == false){
                stop(2); // stop recording
                myView.stopped(); // reset buttons in UI after stopped recording file
            }
    }
}
```

A.5 VmsView.java

```
import javax.microedition.lcdui.*;

public class VmsView extends List implements CommandListener{
Command openCommand;
Command exitCommand;
Image inbox;
Image newVms;
Image outbox;
VMSMidlet mid;

public VmsView(VMSMidlet midlet){
super("VMS application",Choice.IMPLICIT);
mid = midlet;
try{
 inbox = Image.createImage("/images/inbox1.png");
 newVms= Image.createImage("/images/new.png");
 outbox= Image.createImage("/images/outbox.png");
} catch (java.io.IOException ioExc)
        {
//Display alert if an error occurs.......
Alert errorAlert = new Alert("","Error reading image (VmsView) " + ioExc.getMessage(),null,AlertType.ERROR);
errorAlert.setTimeout(Alert.FOREVER);
mid.display.setCurrent(errorAlert);
        }
this.append("  New Voice Message",newVms);
this.append("  Inbox",inbox);
this.append("  Outbox",outbox);
this.setTicker(new Ticker("Select your operation from the menu "));
Command openCommand=new Command("open",Command.ITEM,1);
Command exitCommand=new Command("Exit",Command.EXIT,0);
this.addCommand(openCommand);
this.addCommand(exitCommand);
this.setSelectCommand(openCommand);
this.setCommandListener(this);
}

public void commandAction(Command cmd, Displayable d){

if (cmd.getCommandType()==Command.EXIT){
mid.destroyApp(false);
mid.notifyDestroyed();
}
//      use the public method provided by the midlet to set the appropriate display
else if (cmd.getLabel()=="open"){
int selectedIndex = ((List)d).getSelectedIndex();
mid.setDisplay(selectedIndex);
}
}
}
```

A.6 OutboxView.java

```
import java.io.ByteArrayInputStream;
import java.io.DataInputStream;
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Choice;
import javax.microedition.lcdui.List;
import javax.microedition.rms.RecordEnumeration;
import javax.microedition.rms.RecordStore;
import javax.microedition.rms.RecordStoreNotFoundException;

public class OutboxView extends List{
    RecordStore outboxStore = null;
    RecordEnumeration re=null;
    VMSMidlet mymid;

    public OutboxView(RecordStore obSt, VMSMidlet mid) {
     super("VMS OUTBOX",Choice.IMPLICIT);
     this.setFitPolicy(TEXT_WRAP_ON);// wrap message headers on new lines, show full header
     this.outboxStore = obSt;
     this.mymid = mid;
     this.update();
    }
    public void update(){
        // read nicks from VMSoutbox, and display them in a list;
        try{
         outboxStore =RecordStore.openRecordStore("VMSoutbox",true);
         if(outboxStore.getNumRecords()!=0){
         byte[] byteInputData;
         ByteArrayInputStream inputStream = null;
         DataInputStream inputDataStream;
         re = outboxStore.enumerateRecords(null,null,false);
         this.deleteAll();
         while(re.hasPreviousElement()){
         int temp = re.previousRecordId();//.nextRecordId();
         byteInputData= new byte[outboxStore.getRecordSize(temp)];
         inputStream = new ByteArrayInputStream(byteInputData);
         inputDataStream = new DataInputStream(inputStream);
         outboxStore.getRecord(temp , byteInputData,0);
         this.append(temp + " " + inputDataStream.readUTF(),null);
         }
         inputStream.close();
         }
         outboxStore.closeRecordStore();
        }
        catch (RecordStoreNotFoundException err) {
//          Display alert if an error occurs.......
Alert errorAlert = new Alert("","Error: recordstore outbox not found (OutboxView) " + err.getMessage(),null,AlertType.ERROR);
errorAlert.setTimeout(Alert.FOREVER);
mymid.display.setCurrent(errorAlert);
        }catch(Exception err){
//          Display alert if an error occurs.......
Alert errorAlert = new Alert("","Error: reading VMSoutbox (OutboxView) " + err.getMessage(),null,AlertType.ERROR);
errorAlert.setTimeout(Alert.FOREVER);
mymid.display.setCurrent(errorAlert);
        }
        if(this.size()==0){
this.append("No voice messages",null);
this.removeCommand(mymid.openCommand);
this.removeCommand(mymid.deleteCommand);}
        else{
         this.addCommand(mymid.openCommand);
     this.addCommand(mymid.deleteCommand);
         this.setSelectCommand(mymid.openCommand);
        }
    }
public void openMessage() {
//get the selected index, and extract that message from the RMS and pass it to newVmsView
//int i = this.getSelectedIndex()+1;// adding 1, to start records from 1 and not 0
int i = (this.getString(this.getSelectedIndex())).charAt(0)-48;
String ToNick;
try{
outboxStore =RecordStore.openRecordStore("VMSoutbox",false);
byte[] byteInputData= new byte[outboxStore.getRecordSize(i)];
            ByteArrayInputStream inputStream = new ByteArrayInputStream(byteInputData);
            DataInputStream inputDataStream= new DataInputStream(inputStream);
            outboxStore.getRecord(i , byteInputData,0);
            ToNick = inputDataStream.readUTF();
            byte[] message = new byte[outboxStore.getRecordSize(i)];
            inputDataStream.read(message);
            inputStream.close();
            outboxStore.closeRecordStore();
            mymid.newVms.playMessage(ToNick,message,"outbox");
        }
        catch(Exception err){
//          Display alert if an error occurs.......
Alert errorAlert = new Alert("","Error: in openMessage method (OutboxView) " + err.getMessage(),null,AlertType.ERROR);
errorAlert.setTimeout(Alert.FOREVER);
mymid.display.setCurrent(errorAlert);
        }
}
public void deleteMessage() {
// int i = this.getSelectedIndex()+1;// adding 1, to start records from 1 and not 0
int i = (this.getString(this.getSelectedIndex())).charAt(0)-48;
try{
outboxStore =RecordStore.openRecordStore("VMSoutbox",false);
            outboxStore.deleteRecord(i);
            outboxStore.closeRecordStore();
            this.update();
        }
        catch(Exception err){
//          Display alert if an error occurs.......
Alert errorAlert = new Alert("","Error: deleteMessage method (OutboxView) " + err.getMessage(),null,AlertType.ERROR);
errorAlert.setTimeout(Alert.FOREVER);
mymid.display.setCurrent(errorAlert);
        }
}
}
```

A.7 Communicator.java

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

import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.List;
import javax.microedition.rms.RecordStore;

import peer2me.framework.Framework;
import peer2me.framework.FrameworkSubscriber;
import peer2me.message.Message;
import peer2me.network.bluetooth.Bluetooth;
import peer2me.node.Node;

public class Communicator implements FrameworkSubscriber, CommandListener{
    public Framework framework; // peer2me variable, must be used
List nodeList;
Vector nodesFound;
boolean addShowNodes=false;
public Node currentlySelectedNode =null;
private Alert newmsgAlert;
    VMSMidlet midlet;
    private Command okCommand = new Command("OK",Command.OK,10);
private Command backCommand = new Command("Back",Command.BACK, 0);

    public Communicator(VMSMidlet vmsmid){
     this.midlet = vmsmid;
//      initialize framework
        framework = Framework.getInstance("MyGroup","VMS",new Bluetooth(),true,this);
        framework.setframeworkSubscriber(this);
        framework.initialize();

        nodeList = new List("Nodes", List.IMPLICIT);
        nodesFound = new Vector();
        nodeList.setCommandListener(this);
        nodeList.addCommand(backCommand);
    }
      /**
 * This method is called whenever a new {@link peer2me.node.Node} is discovered
 * @param node the node which is discovered
 */
public void nodeDiscovered(Node node){
        nodesFound.addElement(node);
        }

/**
 * This is called whenever a {@link peer2me.node.Node} is lost or abscent
 * @param node the node that is lost or abscent
 */
public void nodeLost(Node node){
        nodesFound.removeElement(node);
        }

/**
 * This method is called whenever a {@link peer2me.message.Message} is received
 * @param message the message that is received
 */
public void messageReceived(Message message){

String from = message.getSender().toString();
byte[] voiceMsg = (byte[])message.getMessagePart("Mes").getObjectBytes();
// store recieved message in inbox
saveFile(voiceMsg,from);

newmsgAlert = new Alert("New Voice Message", "new voice message recieved from " +
message.getSender().toString()+ "on " + Calendar.DATE,null,AlertType.INFO );
newmsgAlert.setTimeout(Alert.FOREVER);
midlet.display.setCurrent(newmsgAlert);
    }
/**
 * This method is called for every part of a whole {@link peer2me.message.Message}
 * that is received. Is well suited for display progressbars in the GUI etc.
 * @param messageID the id of the message the part belongs to
 * @param part the current part received
 * @param total the total of parts to be received
 */
public void messagePartReceived(String messageID, int part, int total){

}
/**
 * This method is called whenever a search for devices and services is finished.
 *
 */
public void searchCompleted(){
midlet.display.setCurrent(showNodes());
        }
/**
 * returns a List of nodes known by the framework
 */
public List showNodes(){
nodeList.deleteAll();
Node[] nodes = framework.getAllNodes();

for(int i= 0; i< nodes.length; i++)
{ if(nodes[i].getAddress()!= framework.getLocalNode().getAddress())
{
nodeList.append(nodes[i].toString(),null);
}
}
if(nodeList.size()==0){
nodeList.append("No peers found",null);
nodeList.removeCommand(okCommand);
}else {
nodeList.addCommand(okCommand);
}
return nodeList;
}
public void commandAction(Command c, Displayable arg1) {
if (c== okCommand){
currentlySelectedNode = (Node)nodesFound.elementAt(nodeList.getSelectedIndex());
midlet.newVms.recipient.setString(currentlySelectedNode.getNodename());
midlet.display.setCurrent(midlet.newVms);
}
if (c== backCommand){
midlet.display.setCurrent(midlet.newVms);
}
}

//  code for storing sent message in outbox
private void saveFile(byte[] recievedMsg, String from){
// **************open/create record store****************************************
        try{
         midlet.inboxStore =RecordStore.openRecordStore("VMSinbox",true);
        }catch(Exception err){
//         Display alert if an error occurs.......
Alert errorAlert = new Alert("","Error opening inbox (Communicator) " + err.getMessage(),null,AlertType.ERROR);
errorAlert.setTimeout(Alert.FOREVER);
midlet.display.setCurrent(errorAlert);
        }
// **************write data to record store***************************************
        try{
            // data to store...........
            byte[] outputRecord = recievedMsg;
            String nick = from;
            ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
            DataOutputStream outputDataStream = new DataOutputStream(outputStream);
            outputDataStream.writeUTF(nick);
            outputDataStream.write(outputRecord);
            outputDataStream.flush();
            outputRecord= outputStream.toByteArray();
            midlet.inboxStore.addRecord(outputRecord,0,outputRecord.length);
            // release resources
            outputStream.close();
            outputDataStream.close();
        }catch(Exception err){
//         Display alert if an error occurs.......
Alert errorAlert = new Alert("","Error adding record to inbox (Communicator) " + err.getMessage(),null,AlertType.ERROR);
errorAlert.setTimeout(Alert.FOREVER);
midlet.display.setCurrent(errorAlert);
        }
// ****************close record store, release resources*************************
        try{
         midlet.inboxStore.closeRecordStore();
        }catch(Exception err){
//         Display alert if an error occurs.......
Alert errorAlert = new Alert("","Error closing inbox (Communicator) " + err.getMessage(),null,AlertType.ERROR);
errorAlert.setTimeout(Alert.FOREVER);
midlet.display.setCurrent(errorAlert);
        }
        // update outboxView
        midlet.inbox.update();
}
}
```