Helsinki Metropolia University of Applied Sciences
Degree Programme in Information Technology

**James Filbert**

# Developing a Multi-Purpose Chat Application for Mobile Distributed Systems on Android Platform

Bachelor's Thesis. 5 March 2010
Supervisor: Jarkko Vuori, Principal Lecturer
Language Advisor: Taru Sotavalta, Senior Lecturer

**Helsinki Metropolia University of Applied Sciences** **Abstract**

| | |
|---|---|
| Author | James Filbert |
| Title | Developing a multipurpose chat application for mobile distributed systems on android platform |
| Number of Pages<br>Date | 59<br>5 March 2010 |
| Degree Programme | Information Technology |
| Degree | Bachelor of Engineering |
| Supervisor | Jarkko Vuori, Principal Lecturer |

The objective of this project was to design and implement a multi-purpose chat application for mobile distributed systems, which support both instant messaging and file sharing. The chat application is based on peer-to-peer network, which means there is no need for central server for peers to meet and talk. Besides supporting real-time messaging and file sharing, multi-purpose chat application also supports downloading files from the remote web server, and saving the image in the local secure digital card (SD card).

The application client side implementation was done on Eclipse IDE with Android Development Tools (ADT) plugin using Java language and the peerdroid library. A rendezvous peer (a gathering point for peers connected on the JXTA network) was also implemented on Netbeans IDE using Java language.

The results obtained in this project show that it is possible for multiple peers connected on the JXTA network to communicate in real-time manner and share resources with one another. Also users of a multi-purpose chat application were able to download images from the remote web server and save them on an SD card for future sharing with other peers on the network.

| | |
|---|---|
| Keywords | JXTA, peerdroid, android, JXME, P2P |

## Acknowledgments

I am heartily thankful to my supervisor, Dr. Jarkko Vuori, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject.

I would like to express the deepest appreciation to the students and staff of Metropolia UAS for their cooperation during my studies and those who gave me the possibility to complete this project.

Lastly, I offer my regards and blessings to JXTA forum members who responded positively to the questions I sent to the JXTA forum. Also I will not forget friends and relatives who supported me in any respect during the completion of the project.

## Abbreviations

| | |
|---|---|
| CMS | Content Management System |
| DDMS | Dalvik Debug Monitor Server |
| GUI | Graphical User Interface |
| HTTP | Transfer Protocol |
| IDE | Integrated Development environment |
| IM | Instant Messaging |
| Java ME | Java Micro Edition Platform |
| JXME | JXTA for Java ME Platform |
| JXSE | JXTA for Java Standard Edition Platform |
| JXTA | Juxtapose |
| PDA | Portable digital assistant |
| PNG | Portable Network Graphics |
| P2P | Peer to Peer |
| SDK | Software Development Kit |
| SMS | Short Message Service |
| SSL | Secure Sockets Layer |
| TCP/IP | Transmission Control Protocol/Internet Protocol |
| URL | Uniform Resource Locator |

## Contents

# 1 INTRODUCTION

Nowadays, Peer to Peer (P2P) based applications such as file sharing and instant messaging systems have become most popular among the computer users. Computers around the world are linked together and share resources in P2P systems. Every computer on the network is equal in such a way that any computer participating in a P2P is able to access and download resources from other computers in the system.

JXTA is a set of open, generalized peer-to-peer protocols that allow any connected device on the network to communicate and collaborate as peers. JXTA for Java ME, shortly called JXME, is designed to provide P2P compatible functionalities on constrained devices so that the devices could connect to a JXTA network, publish advertisements, discover resources on the network, and share resources with peers discovered. There are two versions of JXME, the *proxy* and *proxyless*. In the proxy version, the proxy do much of the work on behalf of a peer while in the proxyless version a peer becomes a full featured peer, does everything on its own without depending on the proxy. [6]

In 2009, researchers at Dipartimento di Ingegneria dell'Informazione, University of Parma in Italy released a PeerDroid library, the porting for JXME protocols to enable Android application developers to create P2P applications that use the features of JXME system along with Android potential. Since JXME is designed for Java Micro Edition (Java ME) platform, therefore with peerdroid Android applications developers will be able to create P2P applications on Android platform which can interact with mobile terminals. [7]

The objective of this project is to design and develop a multi-purpose chat application for mobile distributed systems using a proxyless version of JXME with peerdroid on Android platform. With this chat application, users would be able to send instant messages and share resources such as photos with other peers connected to the P2P network. Since most of the mobile devices have limited memory capacity, Persistent storage, limited bandwidth, and very critical battery life, I decided to choose Android

Platform as it supports smart phones application development of taking into account the restrictions above.

# 2 ANDROID PLATFORMS, JXTA and PEERDROID

Android [1] is a software stack for mobile devices running on the Linux kernel which includes operating systems, middleware, and key applications. Initially, Android was developed by Android Inc, later purchased by Google and recently by Open Handset Alliance. The Android SDK offers developers the ability to develop extremely rich and innovative applications using the Java language. Android runs on a Linux Kernel and utilizes the Dalvik virtual machine to run the applications. Most of the Android features are already available through other development platforms, which make Android a truly open source development platform, meaning handset makers allowed to use and run it on their devices for free.

## 2.1 The Android Stack Architecture

The Android software stack is made up of four different layers as listed below

- **The Application**

  This is the top layer of the Android software stack which containing built in applications like browsers, maps, calendar and others which are visible to mobile phone users. The applications in this layer are able to run in real time means multiple applications can be done at the same time. [1]

- **The Application Framework**

  The applications in this layer are known as service processes which is full open source. Developers have access to these services but are invisible to mobile phone users. An example usage of one of these services would be an application using the telephony manager to initiate a call. They can be applications supplied by Google or any 3rd party developer. All applications are created equal on the platform meaning 3rd party applications get to use as much of the system resources as in house applications. [1]

- **The Libraries**

These include the surface manager for (compositing windows), Media framework for multimedia files, WebKit (browser engine), Media Codecs like MPEG-4 and MP3, the SQL database SQLite, SGL, SSL, Libc, and OpenGLIES. [1]

- **The Runtime**

Each Android application runs in a separate process, with its own instance of the Dalvik virtual machine. Based on the Java VM, the Dalvik design has been optimized for mobile devices. The Dalvik VM has a small memory footprint. [1]

Figure 1 illustrates how Android software stack layers are arranged.

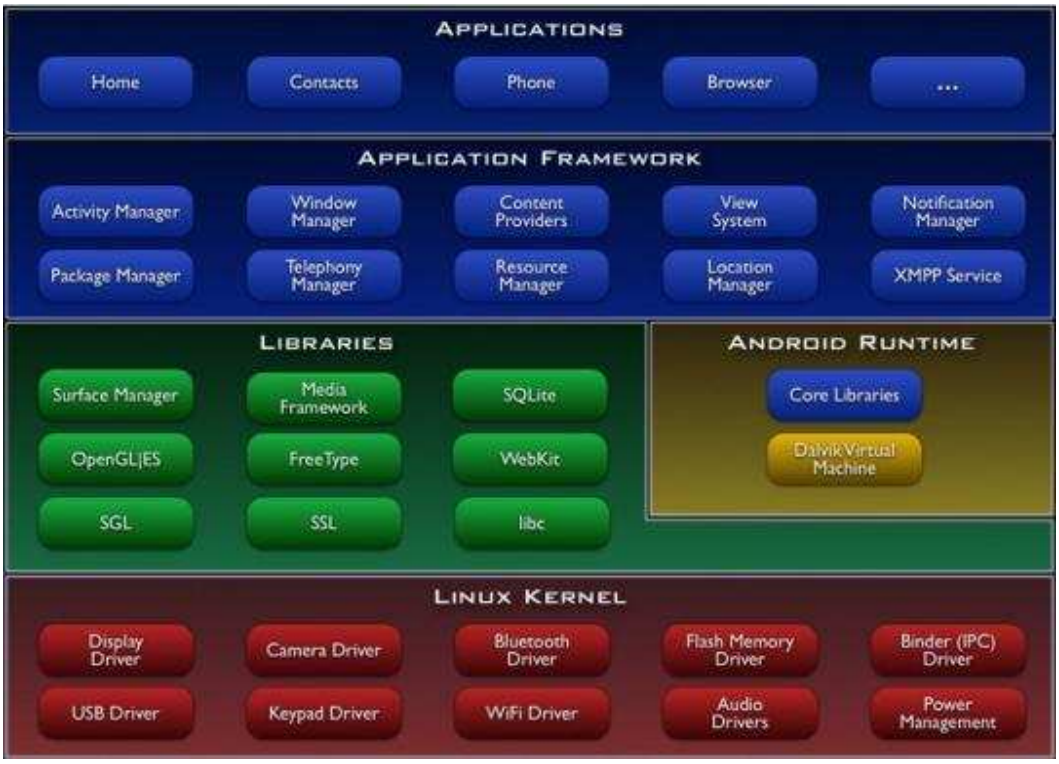



*Figure 1 Android Stack Architecture [1]*

As can be seen from the figure 1, Android software stack is made up with several layers from top to bottom.

## 2.2 Android Devices

By the end of 2009[2], the following devices from different vendors were on market using Android operating system according to Google.

- HTC Dream, Magic, Hero, and Tattoo.
- Samsung Galaxy, BeholdII,Spica
- Motorola Droid ,Calgary
- Acer Liquid A1
- Lenovo
- Sony Ericsson XperiaX10
- Nexus one (Released by Google January 5 2010)


Nexus 1 Motorola droid HTC dream

*Figure 2 Android devices [2]*

As seen in figure 2, devices have big screen size which and can offer advanced capabilities the same as personal computers.

## 2.3 Peer to Peer Architecture

Peer-to-peer networking is a network architecture which allows a group of nodes (peers) to connect with each other and share resources, and any node can operate as either a

server or a client. Hence participants in a P2P network do not need a central server to communicate like the traditional client-server architecture which has existed for many years. Unlike client-server architecture, a P2P network is considered alive even if only one peer is active. The network is unavailable only when no peers are active. [3]

Nowadays the most popular P2P networks file sharing system such as Napster, Ares, Limewire, and Gnutella use decentralized topology, Instant Messaging (ICQ) and distributed computing. [4]

Though peers all have equal status in the network, they do not all necessarily have equal physical capabilities. A P2P network might consist of peers with varying capabilities, from mobile devices to mainframes. A mobile peer might not be able to act as a server due to its intrinsic limitations. The figure 3 illustrates how P2P network is made up. [3]

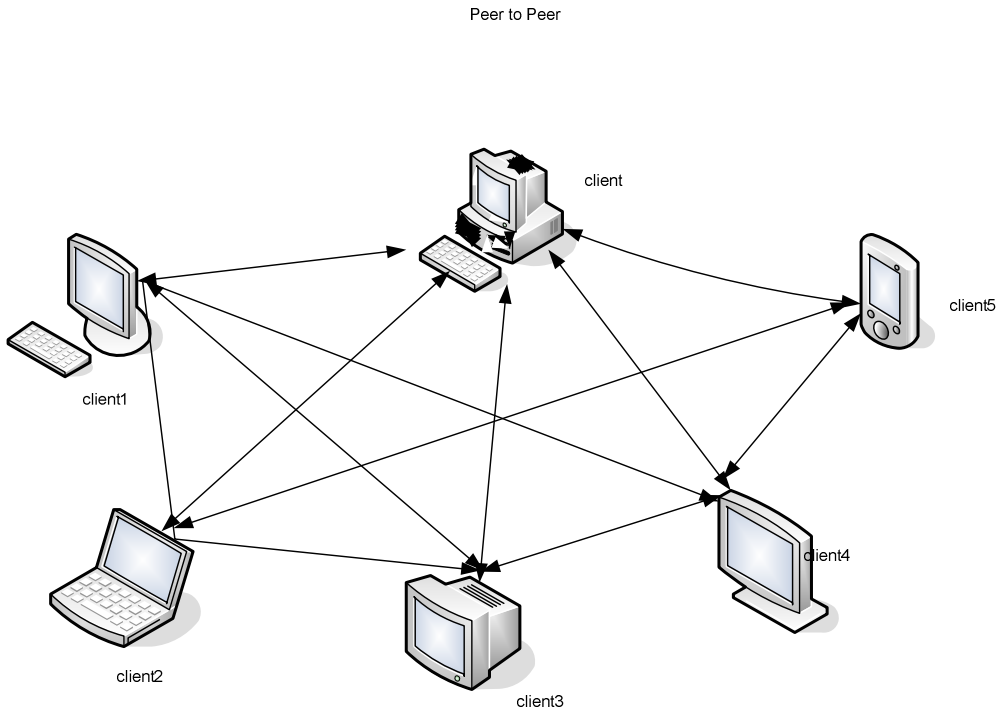



*Figure 3: Peer to Peer architecture*

As seen from the figure 3, peers in P2P network have equal chance in such a way that any peer can act as a server or a client at the same time.

### 2.4 Cient – Server Architecture

Client-server architecture is the oldest technology where a client machine contacts the server when the services are needed. In other words it is called centralized architecture where the whole network depends on a central point. If the central point fails, the entire system will collapse. With no server the network would make no sense.
The procedure is as follows; a client sends a request for a service to a server. The server receives the request and processes the request, and then sends back the response to the client. The client receives the response. Some of the servers existing on the Internet are web servers, mail servers, FTP and so on. [13]

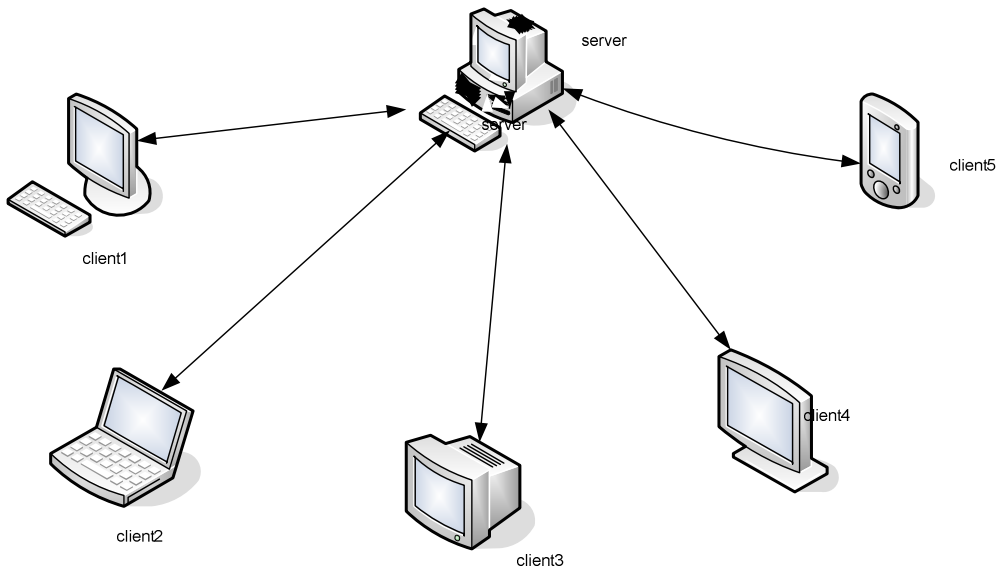



*Figure 4: Client-Server architecture*

Figure 4 illustrates the communication between the server and the client

- Client sends a query *request* to the server

- Server receives a query *request* and process the request
- Server sends the *respond* to a client
- Client receives a *response* as per request.

## 2.5 Project JXTA

### 2.5.1 Overview

JXTA is an open source computing framework for peer-to-peer specification, developed by Sun Microsystems under the direction of Bill Joy and Mike Clary in 2001. The name JXTA is a short hand for *juxtapose* as side by side. The JXTA protocols standardize the manner in which peers

- Discover each other
- Self-organize into peer groups
- Communicate with each other
- Monitor each other
- Advertise and discover network resources
- Platform/operating system independent. [3]

### 2.5.2 JXTA Protocols

To handle JXTA services in a peer to peer system developers of JXTA platform modeled several protocols which facilitate heterogeneous devices to exist and communicate. The following are the protocols in the JXTA platform.

- Peer Resolver Protocol (PRP) – Enable peers to send a generic query and receive a response
- Peer Discover Protocol (PDP) – Used by peers to advertise their own resources.
- Peer Information Protocol (PIP) – Peers use this protocol to obtain status information
- Pipe Binding Protocol (PBP) – Facilitate communication path between peers.
- Peer End Point Protocol (PEP) – Used to find a route from one peer to another.
- Rendezvous Protocol (RVP) – For messages propagation in the network.[4]

Every node in a JXTA network is a peer and each peer has a unique identity called Peer ID which is dynamically bound to its IP or TCP address by the JXTA network, so peers are identified by their IDs rather than IP addresses .[4]

### 2.5.3 Peer Groups

Peers sharing common interest for example cinema lovers, can be grouped together logically to form the peer group according to their taste to discuss and exchange songs etc. Peers are free to create, join and leave the groups. Peers can belong to more than one group at a time. To enable communications between peers which do not have a direct link, a logical communication channel called pipe is used. Every peer will have at least one end point pipe which is dynamically bound to the IP address the peer is using. [3]

## 2.6 JXTA Architecture

JXTA platform [4**]** is divided into three major layers namely The Core layer, The JXTA services, and the JXTA applications. [4] The figure 5 illustrates how the JXTA is made.

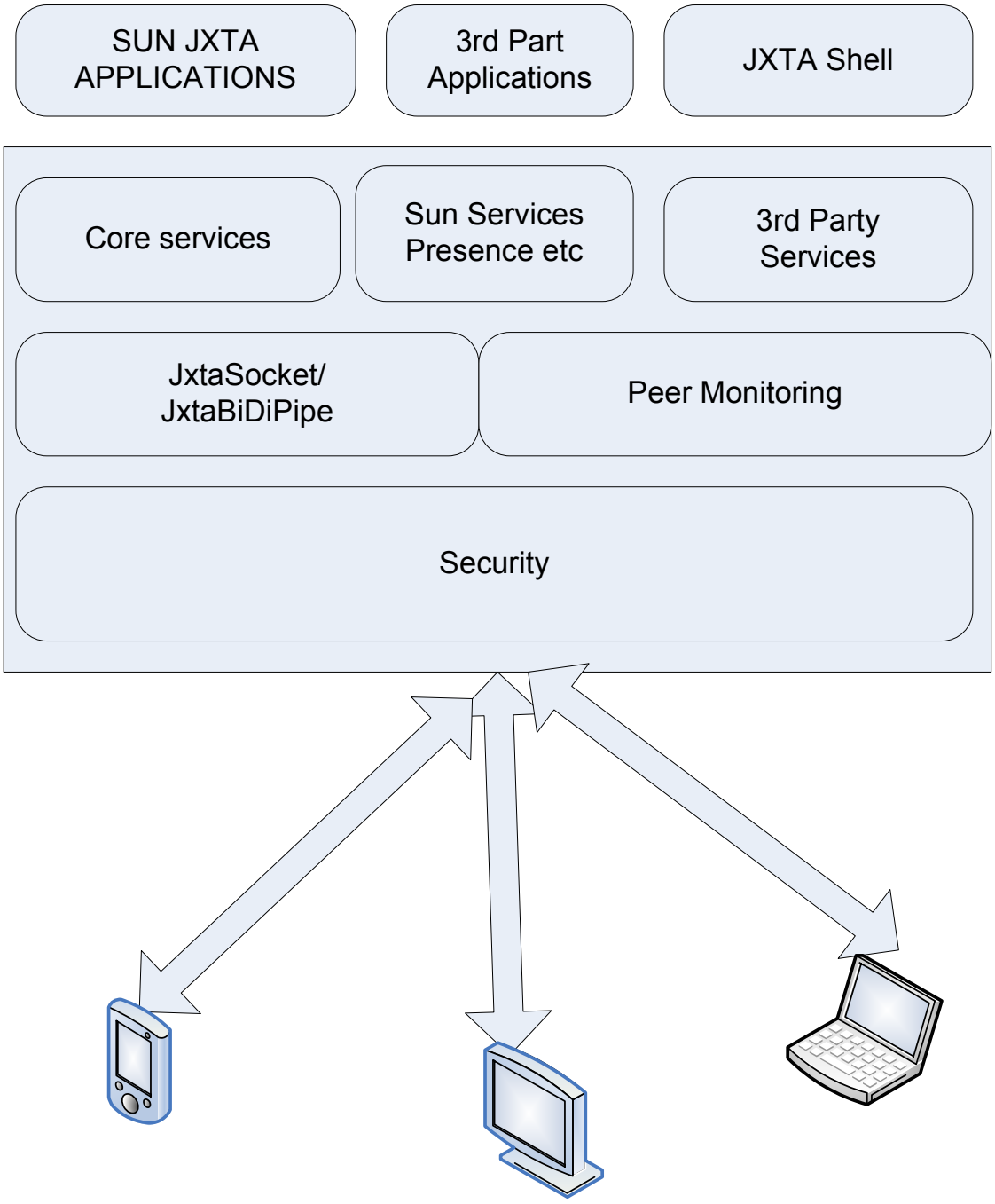



*Figure 5: JXTA architecture*

As seen from the figure 5, JXTA platform is made up of several layers as described in the following subsections.

**The core layer** is where the code for protocols implementation is found in this layer. The following are the elements that ideally would be shared by all P2P solutions.

- Peers
- Peer Groups
- Network Transport (Pipes, Endpoints, Messages)
- Advertisements
- Protocols ( Discovery, Communication , monitoring)
- Security and authentication [5]

**The services layer**: The services in this layer are optional means not necessary needed for P2P network to operate but are common in P2P environment. Such services include searching and indexing, storage system, file sharing, distributed systems, authentication, and PKI (Public Key Infrastructure). [5]

**The application layer** provides common P2P applications that we know such as instant messaging; File sharing, entertainment content management and delivery. [5]

## 2.7 Types of Peers

**Rendezvous peers** are volunteers which act as a meeting point for other peers due to absence of central service such as domain name service. Peers issue discovery queries to a rendezvous peer, and the rendezvous provides information on the peers it is aware of on the network. A rendezvous peer maintains a cache of advertisements, forward discovery requests to help other peers to discover resources and keep a record of other rendezvous peers. Therefore if one knows a rendezvous point of a friend and a friend knows his or hers then one can find each other. [5]

**Relay peers** maintain information about the routes to other peers, routes message to peers and forward messages on behalf of peers that can not directly address another peers. [5]

**Minimal edge peers** are capable of sending and receiving messages but do not cache advertisements or route message to other peers. For example cell phone. [5]
**Full featured edge** peers have all minimal edge peer features plus ability to cache advertisements, but do not forward any discovery requests. [5]

### 2.8 JXTA Pipes

The concept of using pipes in JXTA network is taken from the UNIX operating system and its shell. Pipes use the concept of *endpoint* to indicate the input and output points of communication. Information is put at one end and comes out at the other end. [3]
There are three types of pipes

- **Unicast (point-to-point) –** Connect exactly *two* pipe end points; an input pipe receives messages sent from an output pipe.
- **Propagate pipe** – Connect one output pipe to multiple input pipe.
- **Secure unicast** – One way, secure and unlireliable.

**Advertisement:** When peers and peer groups have services that they want to make known to P2P network, they use advertisement. JXTA advertisement is XML messages meant to publish the availability of specific resources. The figure 6 illustrates how the JXTA advertisement looks like. [5]

```
SocketService Costructor CALL
SocketService PipeADV: <?xml vers...
<!DOCTYPE jxta:PipeAdvertisement>
<jxta:PipeAdvertisement xmlns:jxt...
    <Id>
        urn:jxta:uuid-59616261646...
    </Id>
    <Type>
        JxtaPropagate
    </Type>
    <Name>
        test
    </Name>
</jxta:PipeAdvertisement>
```

*Figure 2.6: Pipe advertisements*

As seen from the figure 6 the *Id* stands for unique id of the pipe in UUID format. The *Type* identifying the pipe type whether is a propagate pipe or a unicast pipe. The *Name* is for pipe name.

**Discovery**: Before they can exchange contents in P2P network, peers must discovery each other. To search for peers in the same group, peers use discovery protocol. For peers runs in different network should contact the rendezvous and request that it perform search. The rendezvous cache all the peers it comes into contact with. Always discovery depends on the number or rendezvous in the network, large discovery takes place if there more rendezvous peers on the network. [16]

### 2.9 Project JXME (JXTA for Java ME)

The idea behind to create a project JXME popular JXTA for Java ME was to provide JXTA compatible functionalities on constrained devices (mobile phones, pagers and PDAs ) so that devices could connect to JXTA networks. [6]
The following are two types of JXME. JXME proxy and JXME proxyless.

**JXME proxy** : In JXME *proxy* version , peers talks to JXTA relay ( a message relaying peer) ,which in turn bears most of the message processing (XML authoring for advertisements, sending, search messages across the JXTA network and so forth) and relaying burden. The situation is quite similar to the client-server architecture. A peer sends a request to the JXTA relay for certain service (creating pipe/groups etc), the JXTA relay process the request and send back the response. The figure 7 below illustrates this procedure.

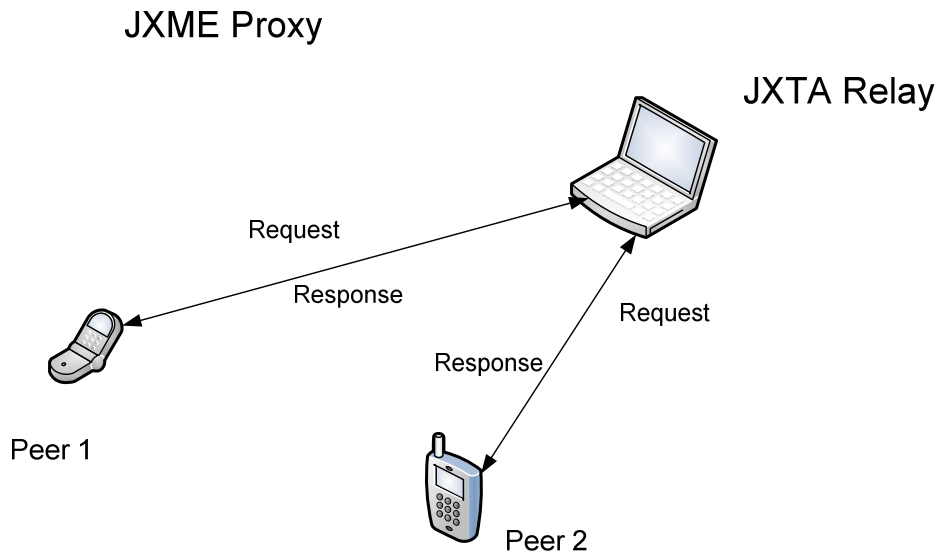



Figure .7 JXME proxy

**JXME proxyless**: The case is different for peers using JXME *proxyless*. In JXME *proxyless* peers became full featured peer. A full featured peer just needs a rendezvous (meeting point) for peers to discovery each other and share resources. Peers in proxyless version do not depend on proxy for pipes and groups creation, relaying messages etc. A mobile device is now able to:

- Describe and publish advertisements
- Discover network resources
- Establish direct and virtual multicast connections to other nodes
- Use a datagram like asynchronous bidirectional JxtaBiDiPipe [15]

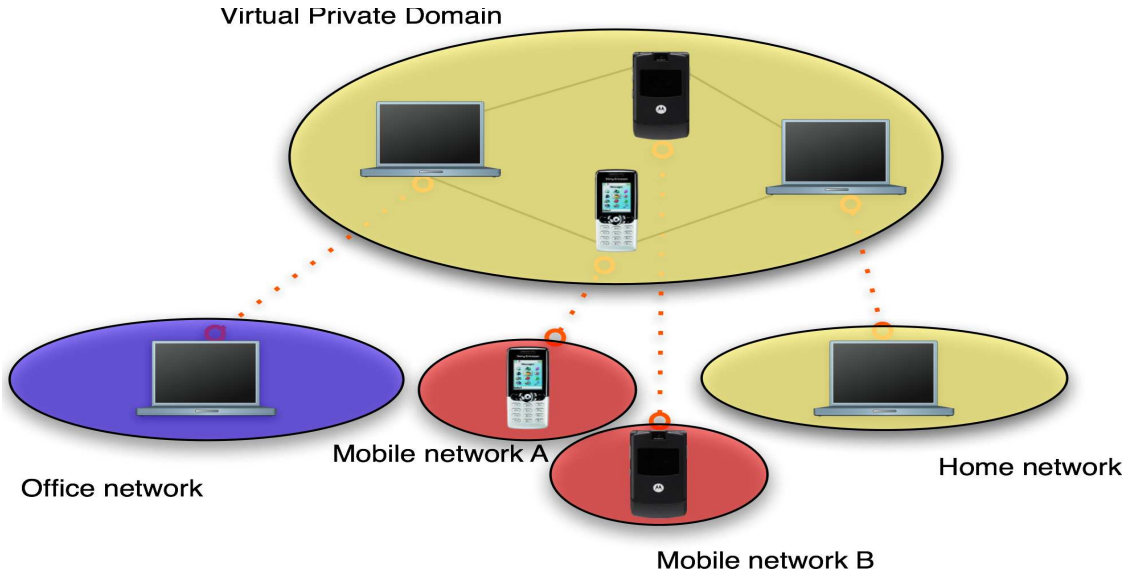



*Figure .8: JXME proxyless* [15]

Figure 8 illustrates how the JXME proxyless network is work.

**Peerdroid** is the porting of JXME protocol to Android platform developed by researchers in Italy in 2009 to enable Android application developers to create P2P applications that uses the features of JXTA system along with Android potential. PeerDroid support both HTTP and TCP/IP protocols. Since JXME is designed for P2P applications on J2ME platform, Peerdroid become a solution for P2P applications on Android platform.

**JXTA CMS** (Content Management Service): To retrieve and share files contents within a peer group a Content management Service (CMS) is used. [9] CMS allow peers to search for files within a group and allow downloading for available files from remote peers. All this is possible through advertisements. Peers must advertise file contents to be shared to make it known in the peer group.

**AndExplorer** is a file manager for Android devices. With AndExplorer users are able to browse files and folders stored on the device and SD card. Files and folders are sorted according to size, name and the date of arrival.

# 3 CHAT APPLICATIONS ON MARKET

Before developing my chat application I did a research on various chat applications available on the market at the same time taking into consideration the technology I'm going to use. At the beginning I considered to use a proxied version of JXME but after doing a research and talk with some JXTA forum members I realize that JXME *proxy version* is outdated technology, and no longer supported. So I decided to use JXME *proxyless* version where a mobile device became a powerful peer performing all work on its own without relaying on a proxy.

The idea behind was to create a chat application for Java micro edition platform. Those devices with limited memory capacity, screen size etc, but due to lack of tutorials for JXME *proxyless* version I decided to use a Peerdroid which is the porting of JXME protocols for Android platform. Because Android platform support next generation mobile devices, with no restrictions as those of Java ME platform above, I found is much better for P2P file sharing applications. The following are some of the Android chat applications I found on the market.

## 3.1 Bluetooth Chat on Android

Bluetooth chat application allows mobile devices connected over Bluetooth to carry out two way chat. The application does the following task

- Scanning for Bluetooth devices
- Sharing of file over Bluetooth
- Establishing RFCOMM channels [1]

## 3.2 Android Chat

Android Chat is a location-aware chat client for Android platform which support location-aware services and a client for Google Android platform. It allows users to connect to centralized server and discovery other chatting users. It was initially designed for social purposes

- At s sports event chatting about the game
- For business promotion  purposes
- For fishing and hunting trips
- For matchmaking purposes [8]

The Android chat application has the following features.

- **Switch between Windows –** Users of this are able to switch from one window to another. From the main menus that pop up, hit the open window button to switch to the window you want to.
- **Creating channels –** Users are able to create their own chatting channels and join to the channel created.
- **Send a private message –** Users are able to switch from a group chat to private chatting.[8]

### 3.3 IMS SIP Client on Android

IMS SIP is a simple chat client for SIP instant messages over IMS run on Android platform. Before any operation a user must sign in the application using his/her email address. IMS SIP client has the following features

- Retrieving chat history at the end of conservation
- Support multiple buddies
- Voice signal for incoming message. [11]

### 3.4 Peerdroid Sample Chat

Peerdroid sample chat application is developed as a demo by researchers who created the peerdroid library to illustrate how peerdroid is capable for P2P applications .The application has a simple GUI to show incoming messages, network status and available peers. [9]

# 4 MULTI-PURPOSE CHAT APPLICATION

## 4.1 Overview

Multi-purpose chat application is based on JXME *proxyless* version ported to Peerdroid. It allows users to send asynchronous messages, and enable sharing image files with other peers on the JXTA world. The application is designed for Android mobile phone users.

The application first connects to the JXTA world, and then discovery the peers already connected to the network and the resources available. The peer should also publish/ advertise the resources it has.

**Motivation**: To develop an application based on new JXME *proxyless* version with Peerdroid for next generation mobile devices. The smart phones which have greater multimedia content capturing capabilities (voice, video, photo etc). The idea of developing multipurpose chat application come from researching on existing chat applications, identifying the positive and negative features also friends ideas to find out what features do they need as they are the end users.

**Features**: Multipurpose chat application has a *editText* field which allow a user to write the message content then press the *send* button to send message content out the unicast bi-directional pipe. The *editText* is designed to handle a certain amount of characters. When number of characters exceeds the buffer size, an exception occurs.Along with message is the name of the sender. The name is included to show the receiver from whom the message is from and the time of arrival. The Incoming messages will be displayed in scrollable form and can be retrieved later as chat history. When a peer received a message can reply back. First should write the message then click *send*. A user will be able to download image from the remote web server and share with other users of the system. The downloaded image files could be saved in a local SD card as compressed files and retrieved later for the future uses.

### 4.4 The Uniqueness of the Application

As it seems my mobile chat application has many features in common as features available on chat applications above but still all applications already on market use a common communication technology (client-server) to work while my application is based on P2P technology. The Bluetooth Chat uses Bluetooth to transfer data /instant messages between devices. The IMS SIP chat uses IMS SIP protocol for communication.

In my case peer does not need a central server to discovery and talk to each other, they just needs a rendezvous peer to enable discovery and make advertisement for resources they have. Multi-purpose chat application downloads files directly from remote web server then share with other peers online. When a peer has a content to share then must retrieve it from the secure digital card (sdcard*)* and send the compressed file to other peers in the JXTA network.

The device vibrates when the message received. Unfortunately one can't notice a vibration with virtual device (emulator) because at the moment emulator does not support vibration but it should vibrate on real device (mobile phone).

### 4.3 Comparison to Peerdroid Sample Chat

Peerdroid sample chat is neither capable for file sharing nor able to switch from screen to screen as it is not implemented for that. Multipurpose chat is capable for file sharing while peerdroid sample doesn't at the same time different in communication channels. Peerdroid sample chat uses a *JxtaSocket* while multi-purpose chat uses a bi-directional pipe (JxtaBiDiPipe).

Multi-purpose chat receives messages with vibration in scrollable format also application is implemented to allow users to log out when they wish to do so, but peerdoid sample doesn't. Multi-purpose chat allows users to download files from remote web server and save to local SD card. The last thing is different approach in graphical user interface approach in both cases.

# 5 DEVELOPMENT ENVIRONMENT, TOOLS, AND SOFTWARE

## 5.1 Eclipse IDE Installation

The Android SDK support several different integrated development environments (IDEs) but the eclipse IDE (Galileo 3.5) with the Android development tool ADT plug in was used for development of this application. Eclipse IDE is highly recommended approach to Android development. The IDE was downloaded from the download page of the eclipse.org. Downloaded zip folder was unpacked to a known directory. Then JDK 6 or higher and JRE was installed too for good performance of the development environment

## 5.2 Downloading and Installing the SDK Starter Package

In this work I used a new version of the SDK Android 2.0.1 downloaded from the Android developer site. The SDK zip file was unpacked to a known location which was then added to the PATH of SDK. Next step was installation of SDK components. For eclipse with ADT just select Window>Android SDK and AVD manager
Select available components, then install selected

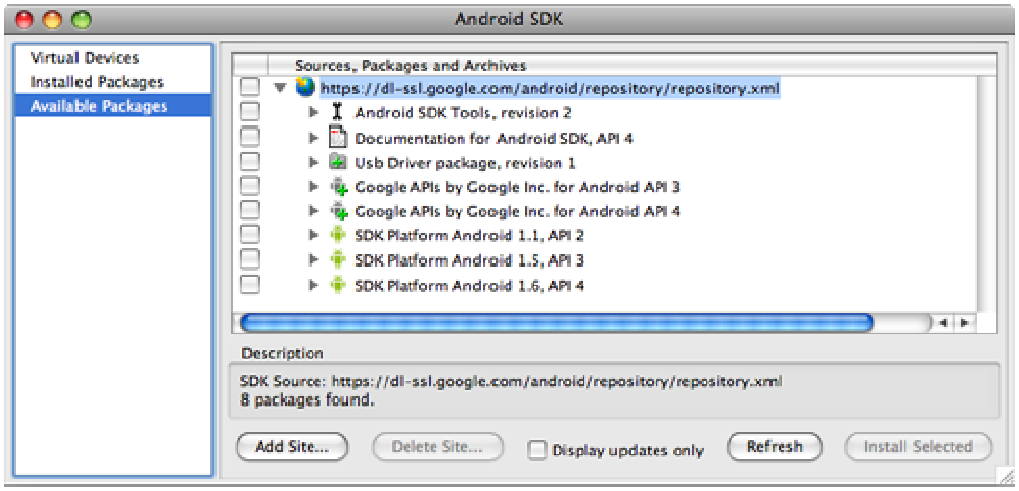



*Figure 9 Installing SDK starter package*

Figure 9 shows the available SDK starter package for download.

### 5.3 Installation of the ADT Plugin for Eclipse.

Android Development Tool (ADT) is designed to give a powerful environment for integrated environment for application development. The ADT extends the capabilities of eclipse to create an application user interface, debugging application, and adding components based on the Android Framework API. [1]

Once eclipse is installed as described above ADT plug in should be installed in respect to eclipse environment. Start eclipse then select help>Software updates>available software. Click add site then add this URL in location https://dl-ssl.google.com/android/eclipse/. Back in the available software view you should see plugin listed by URL with Android DDMS and Android development tools then next.
Read and accept the license agreement then install. Finally restart IDE.

### 5.4 Importing Peerdroid

Peerdroid is piece of software library available for free on Google projects websites. A developer should download and save the library file in a certain directory. The Importation of this library to the project is done from the eclipse IDE, file->properties then Java Build Path. Add external Jars. Add peerdroid from the directory you saved.

### 5.5 JXTA Shell as a Rendezvous

As I section 2.5, in order for peers to communicate, a rendezvous is needed to facilitate the discovery in the JXTA system. There are two options to make a rendezvous peer. The first option is to use a JXTA shell. The shell is available for download on JXTA website then should be installed in the computer with public IP; in this project I used a JXSE-shell-20080909 nightly version. The second approach is to implement a rendezvous peer in any suitable IDE using java language. [4]

The next step was to configure a JXTA shell to act as a rendezvous. My intention was to use public ready configured rendezvous peers for testing purpose but during the development of this application Sun Microsystems router was down and unreachable.

After finishing a rendezvous setup I tested to see if it was active for peers to discover each other advertise and communicate. This was made possible by typing some useful commands on the shell as you can see in the figure 10 and 11 below. To discover peers connected to rendezvous I used '*peers*' command and '*rdvstatus*' to check the status of rendezvous.

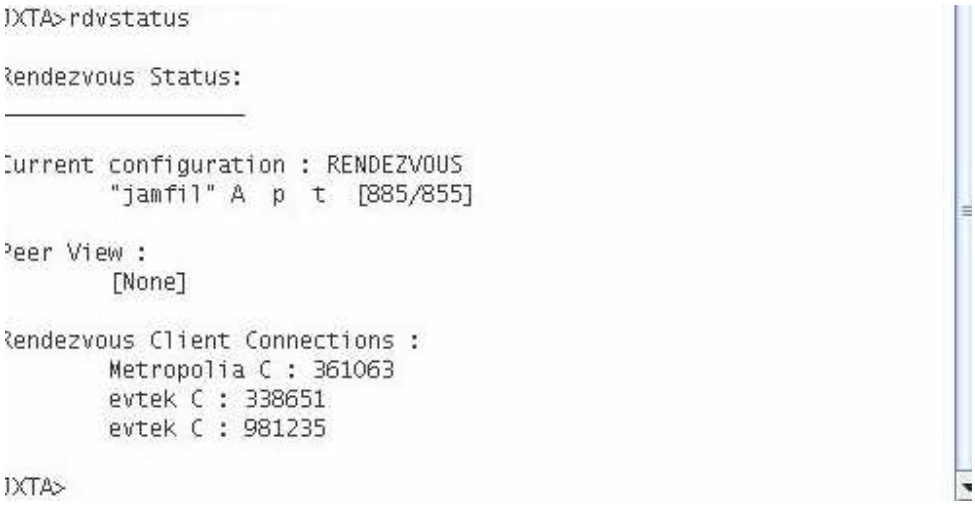



*Figure 10. JXTA shell: Rendezvous testing*

Figure 10 shows the status of the rendezvous peer when clients connected.


*Figure 11. JXTA shell: Peers View*

Figure 11 display the names of peers connected to the JXTA shell

### 5.6 Implementing the Rendezvous Peer on IDE

Instead of using a JXTA shell as a rendezvous, another approach which is better, is implementing a rendezvous on any suitable IDE using java language. With this kind of rendezvous one doesn't need to clear the local cache for previous advertisement. Advertisement history is deleted automatically every time the IDE started. But if a developer is using a JXTA shell as a rendezvous then must clear the local cache by deleting the CM directory otherwise the old peer advertisements will be discovered. In this project a rendezvous was developed on *Netbeans* IDE 6.5 using java language. [17]

The figure 12 below shows how this rendezvous works and the peer-rendezvous connection.The sample code for this kind of rendezvous peer is shown on appendix A


*Figure 12. Connections to Rendezvous Peer*

Connection to the rendezvous peer as can be seen from figure 12. The Ip address of the client connected to the rendevous peer is shown as well as the time the peer get connected.

### 5.7 Implementation, Testing and Evaluation of the Result.

Application will be developed based on JXME *proxyless* version using peerdroid library on Android platform. This is somehow far from a normal chat familiar to many people because apart from chat, a user will be able to download photos from a remote web server

and share with his/her friends. I have tried to make the application a user friendly, so no need for a user to have any extra knowledge to use it. Multiple screens/dialogs will be made to enable a user to switch from one dialog window to another.

When the application is ready, it will be tested on different emulators run on different computers to see if the user could be able to chat /share resources with other peers connected to Rendezvous. If I will succeed to communicate using emulators then final test will be on the real device mobile phone which is the target of my project.

# 6 DESIGNING AND IMPLEMENTING OF THE SYSTEM

## 6.1 Graphical User Interface

User interface should be taken in mind when a software developer creates an application. A friendly user interface was the first thing I considered when doing this project. The Android platform enables applications developers to create GUI easily using XML layout. The XML files are stored in the layout directory of the project. This chat is designed to have several screens/dialogs where a user will be able switch from one screen to another according to what task he/she wants to perform. Here below are dialogs existing in this application.

The **greetings screen**: This is the first screen when the application starts. The greetings dialog window has a chat image indicating it's a chat application together with progress bar indicating connection to the rendezvous. The dialog window dismiss when the application gets connected to JXTA network. Connecting to rendezvous peer basically means that the rendezvous peer is added to the querying peer's list of known rendezvous peers. There are two ways to connect to rendezvous peer: sending raw connects advertisements or via a method of *RendezvousServeice*. If the rendezvous peer is unreachable then the greetings dialog will not disappear, the progress bar will run forever.


*Figure13. Screenshot: Greetings Screen*

The figure 13 illustrate the first screen appeared when the application is launched.

The **main screen**: Once the application gets connected to the rendezvous, the welcome window dismisses and the main window appears. The screen has several buttons and the *editText* field where a user writes the text message. After finishing writing the text on the text field a user click button "*send*" to send the message content to the unicast pipe. Apart from send button there are also folder button which when clicked it display the list of saved files. The button *image* switches the window to download screen. The *sfile* button for sending files to remote peers. The clear button clears the conversation text on the screen.

When a user clicks a main menu, three options appear. The first option is for *buddyList,* if clicked the buddy List dialog window which display peers connected on the JXTA network appears. The second option is the button "*download"* icon. When pressed a download dialog window appear. Here a user will be able to write the URL of the remote web server where he/she can download image file. The last option button is "*Quit"* button.


*Figure 14. Screenshot: main screen*

Figure 14 illustrate the main screen appeared when the application get connected to the rendezvous peer.

The **FileImage screen**: The screen contains a *editText* field where a user will be able to enter the photo URL to download image from a remote webserver.User type the URL on

the *editText* field, then click the load button to download the image and display it on the screen as a bitmap.


*Figure 15. Screenshot: File image (Bitmap)*

The figure 15 illustrates the image downloaded from the remote web server and display on the screen as a bitmap

**Saving the image file**: Downloaded image file should be compressed, saved and renamed in local secure digital card (SD card*)* before can be sent to other peers. The reason is that Android stores pictures and video on the SD card. The *SD card* is created in the folder where android plugin files are located. From the command prompt go to the

tools directory then type command *mksdcard* [specify the size and the name of the SD card]. To browse the compressed files stored on the SD *card*, *AndExplorer* file manager was used. [10]


*Figure 16. SD card creations*

Figure 16 shows how to create the local storage device (sdcard) from a command line.

The **disconnect screen**: When a user wants to log off the system then he or she clicks the Icon button *Quit* from the main screen, a disconnect dialog window will appear. The dialog asks the user if he or she really wants to sign out, and if the answer is *yes* the disconnect dialog disappear then *disconnect* method is called. If the answer is *no*, the application goes back to the main screen.


*Figure 17. Screenshot: logout dialog*

As can be seen in the figure 17, the application asks the user if he or she want to log off the system.

The **buddylist screen**: Peers connected to the rendezvous are listed on buddylist screen. The first peer to connect on JXTA network will be the first and vice versa. The screen has a chat buddy icon just for user interface decoration.


*Figure 18. Screenshot: buddylist dialog*

As seen on the figure 18, discovered peers connected to the rendezvous peer are display on buddylist dialog.

### 6.2 Class Diagram

The prototypes and the implementation of the system consist of several classes and functions which perform different tasks independently or sometimes in collaboration. Classes of the system, their inter-relationship and the methods and attributes of the classes are illustrated in the following figures though not in a good order due to lack of spacing.

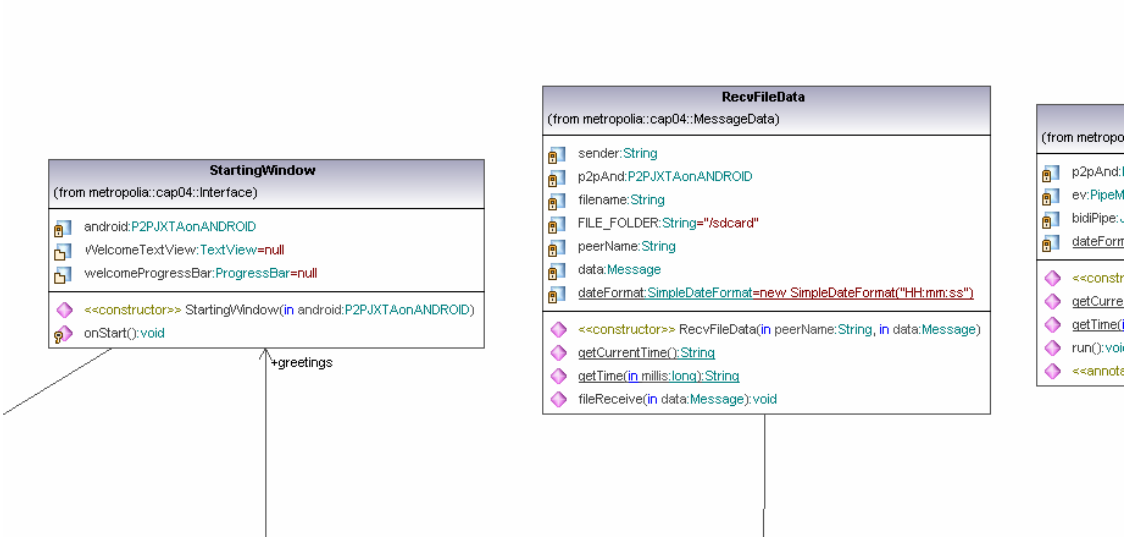
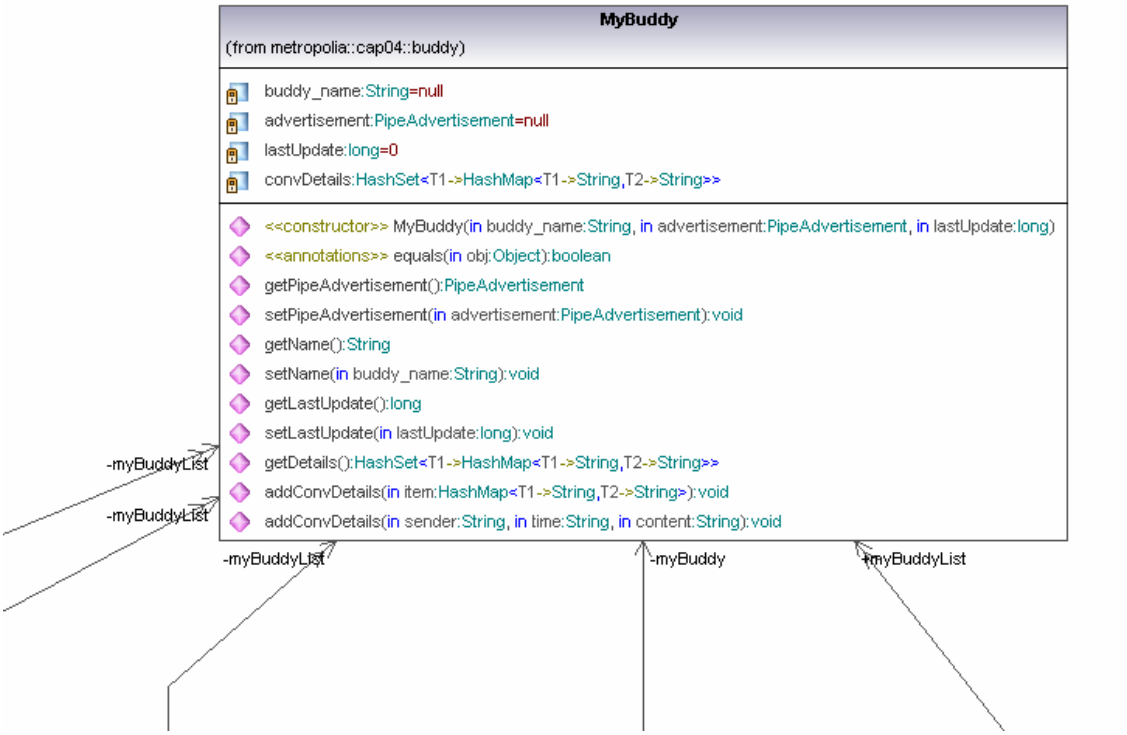



*Figure 19. Class diagram part 1*

Figure 19 illustrates the attributes and methods of two classes StartingWindow and RecvfileData and their interactions with other classes.



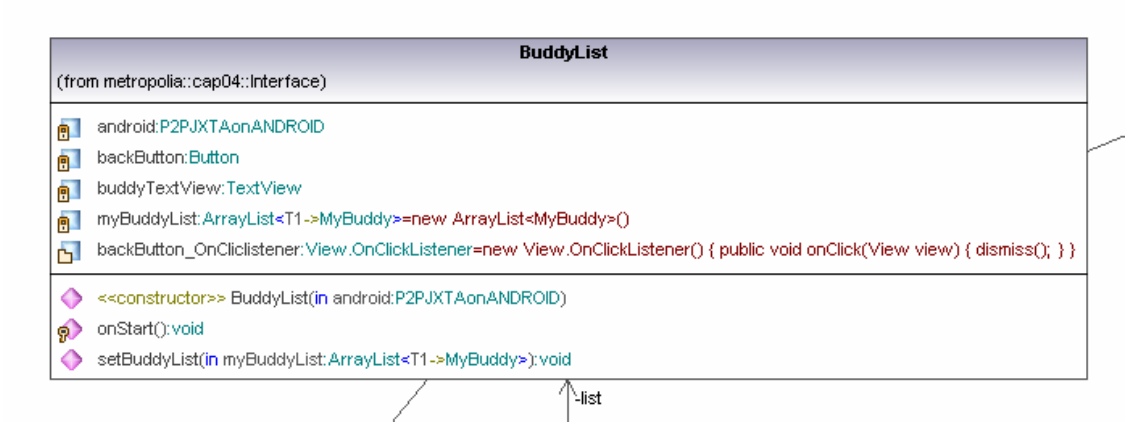
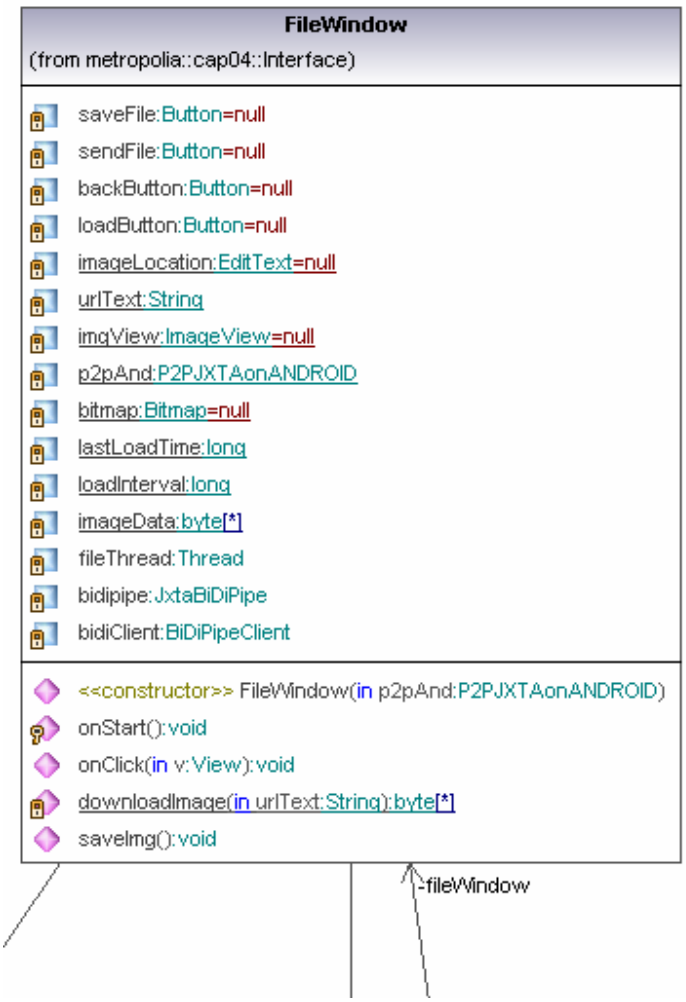



*Figure 20. Class diagram part 2*

Figure 20 Illustrates the methods and attributes for classes BuddyList and MyBuddy as part of the system.



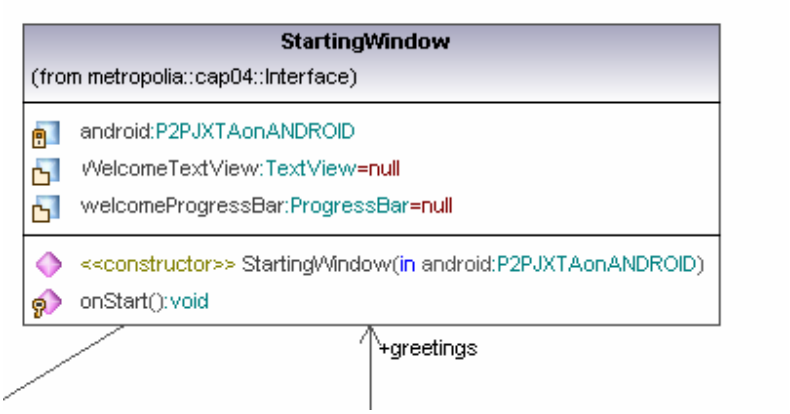



*Figure 22. Class diagram part 3*

Figure 22 illustrates the attributes and methods for two classes the *fileWindow* class and the *StartingWindow* class .

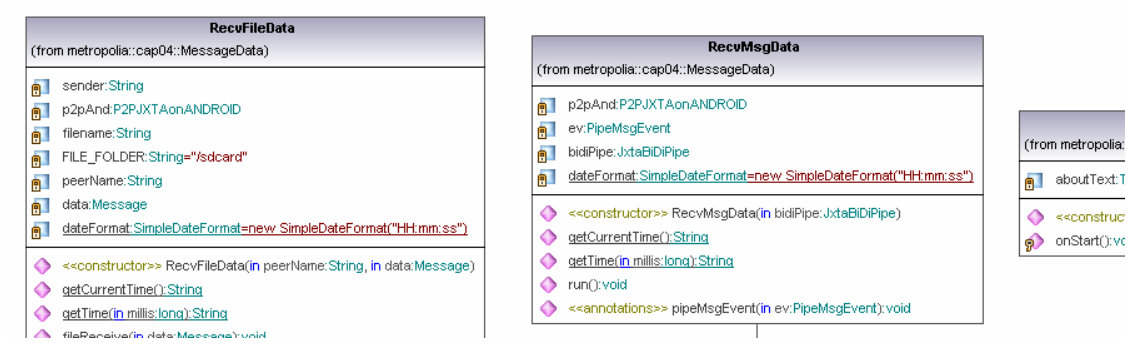



*Figure23. Class diagram part3*

Figure 23 illustrates methods and attributes of *RecvFileData* and *RecvMsgData* classes.

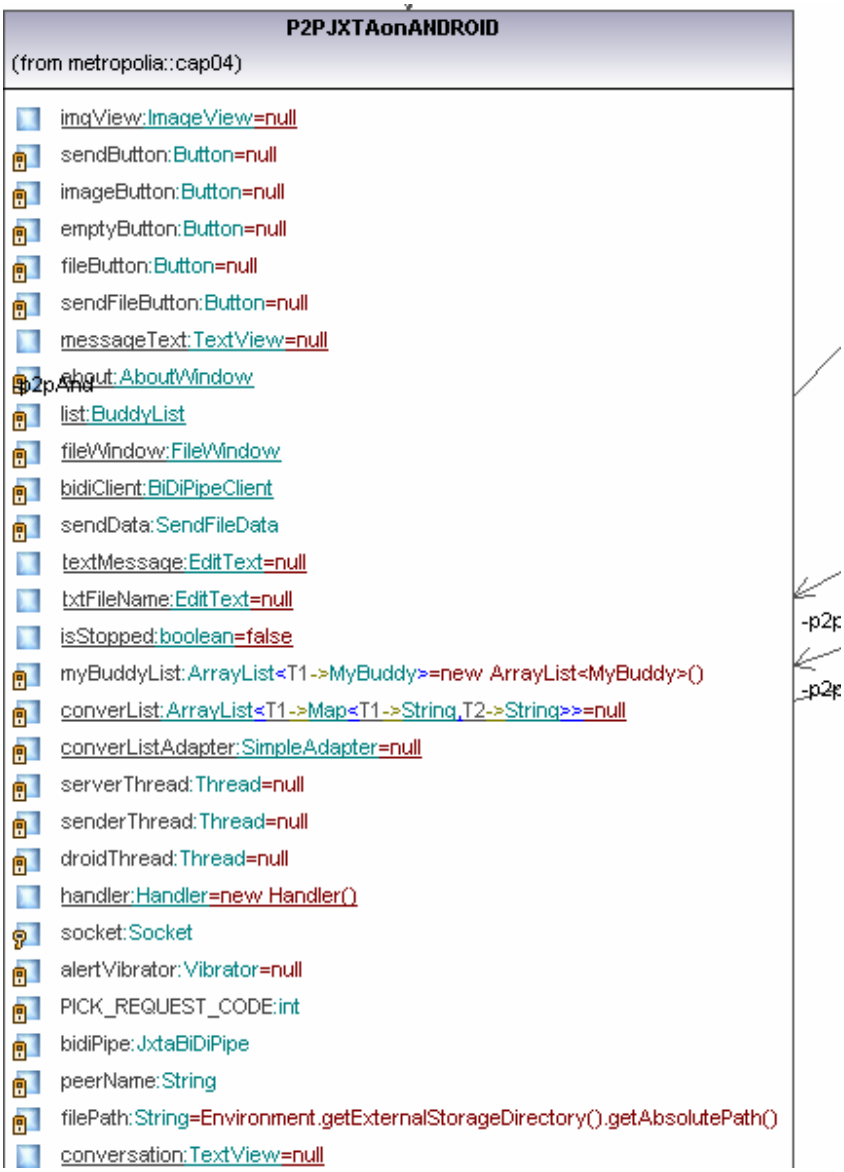



*Figure 24. Class diagram part 4*

The main activity of the system as illustrated in the figure 24.

### 6.4 GUI Classes

The **AboutWindow** is a dialog window which gives the information about the chat system, why it is developed and the developer name. The dialog has one button *backButton.*


*Figure 25. Screenshot: About dialog*

As seen on the figure 25, the dialog shows what for the application was created.

The **BuddyList class** is a dialog window which shows the list of peers discovered from the rendezvous peer. All peers connected to the rendezvous peer will be listed in this dialog.

The **FileWindow** is a dialog window which allows a user of the application to download image files from remote web server. The window has *editText* field where a user can enter the URL of the web server and several buttons. The *sendButton* for sending downloaded image, the *saveButton* which compress the downloaded image, rename and

save the image file. The *backButton* calls the method *dismiss()* then go back to main chat dialog.

The **WelcomeDialog**: This is the first dialog window when the application starts. The dialog dismiss when the applications get connected to the rendezvous peer.

### 6.5 JXTA Platforms

The **BIDiPipeClient class** is the main class which associates the peer with the JXTA platform before any operation can be invoked in the system, which means it handles all communication with the rendezvous peer. The class has several methods from peer configuration to peer advertisement, publishing peer advertisements and discovery methods for advertisements. The class implements *DiscoveryListener*, to enable peer discovery in the network.

The **BiDiPipeServer class** is the class where communication channels (JxtaBiDiPipe) are created and published to the network. The *JxtaBiDiPipe* created is the one for sending and receiving data in the P2P network. I decided to choose this type of socket rather than normal pipes because sockets are bidirectional pipe, reliable and secure than pipes. When one decides to use normal pipes then must create two pipes, *OutputPipe* for outgoing data and the InputPipe for incoming data

The **MsgProcessing class** is the class handles the messages processing to the destination. The class implements the *Runnable* interface. The *run* method keeps the process running for listening outgoing messages and data as long as peers are connected on the rendezvous. The class has a method for sending messages which bears the name of the sender and the content of the text messages.

The **RecvMsgData class** is the class responsible for polling the incoming messages. The class extends thread class .The RecvMsgDataclass has a method which returns the current

time for incoming messages in Hours: Minutes: seconds' format.The sample code for sending and receiving message elements is illustrated in appendix B.

The **DiscoveryController class** is the same as *MessageSender* class and *Message receiver* class. The discovery controller also implements *runnable* interface listening for advertisement published by *BiDiPipeClient*. The class is listening to both local and remote advertisements.

# 7 PERFORMANCE AND RESULTS ANALYSIS

## 7.1 Overview

The objective of this project was to design and implement a chat application based on the P2P technology, which allows sending and receiving instant messaging and sharing image files among peers connected to JXTA network. The application client side was implemented on the Android platform using the Java language and the *peerdroid* library file downloaded from *peerdroid* project site, while the rendezvous was implemented on *Netbeans* IDE 6.5 using a Java language. [17]

## 7.2 Connection to Rendezvous Peer

This was the first part of the application before other operation of the system invoked. The idea behind connecting peers to the rendezvous is to facilitate peer advertisements and discovery before they can share resources. The connection worked as expected though at the beginning it was difficult but after certain trials finally I succeeded to connect. The reason I did not connect is that in the Android security model, all applications have no permission to the internet by default. Therefore I added to AndroidManifest.xml file the following line of code <uses-permission android :name =”android.permission.INTERNET”/>

## 7.3 Sending and Receiving Messages with Vibration

A user was able to write the message in the *editText* field then click a *send* button, the message successful sent through the unicast pipe. The message has 3 elements. The name of sender, time in Hh:Mm:Ss format and the message content. This part was first tested on emulators run on different PCs connected to a rendezvous peer. Message was successfully sent through the socket. The vibration method was well implemented to notify the arrival of the message but unfortunately at the moment the Android virtual device (emulator) did not support vibration, so it was not possible to notice the vibration.

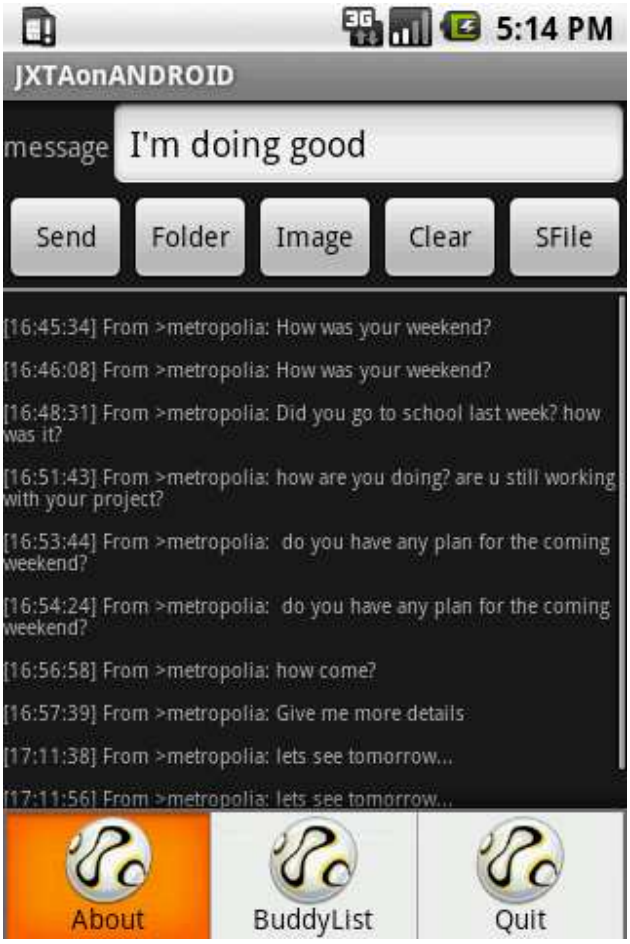



*Figure 26. Screenshot: Instant messaging in scrollable form*

As it seen on the figure 26, the received messages are in scroll form showing the sender name and the time of arrival.

### 7.4 Sharing Image Files

The idea was to have an application which enables downloading image files from a remote web server. To make this possible a user writes the website URL where an image is located then clicks the button "*load*". When the image is downloaded, it is displayed on the *imageDataScreen* as a bitmap, and then the file must be compressed and saved on the SD card. The test went perfectly as required, and I was able to download images from

various internet sources and save them to the *SD card*. Also sending and receiving files in P2P style was fine as expected. The received files were stored in the SD card the same as those downloaded from the remote web servers.

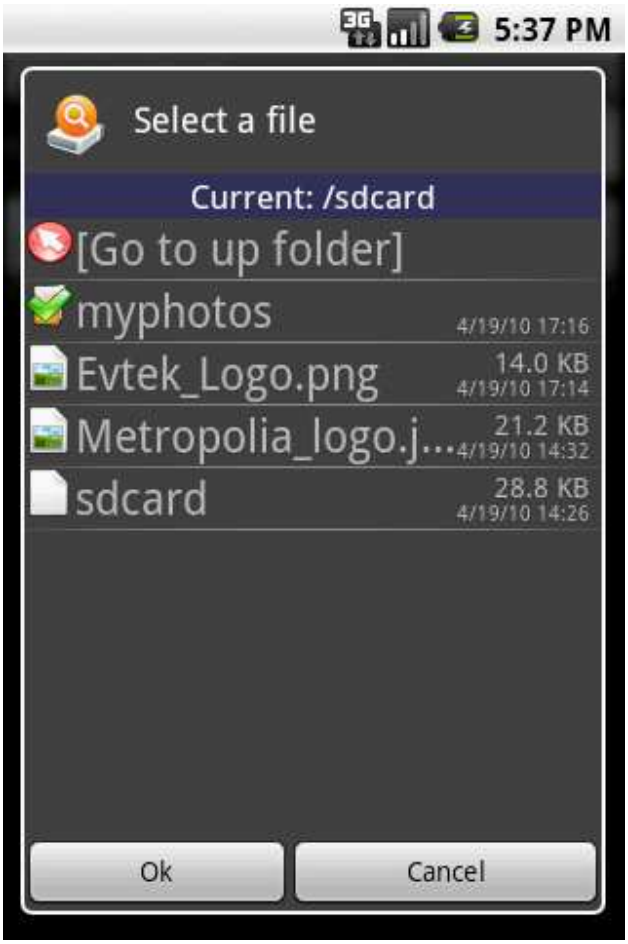



*Figure 27. Screenshot: File list*

The figure 27 shows the listing of the downloaded /received image files from various sources.

## 7.5 Limitations

P2P architectures are naturally suitable for implementing asynchronous messaging and file sharing systems. Developing P2P applications for mobile devices on the Android platform is a new and promising research area, rising with a new mobile technology.

JXTA is a highly complex P2P framework with many protocols and concepts. For newbie applications developer needs both resources and enough time to deal with this platform. Since it is an open-source project, changes on platform source codes occurs in a very short period of time  but unfortunately the JXTA platforms lacks the tutorial guide for mobile applications development. The JXTA tutorial guide released in 2007 just discusses the P2P applications development on personal computers (JXSE), so it is very difficult for a P2P mobile applications developer to get started.

# 8 CONCLUSION

The goal of this project was to design and develop a multi-purpose chat application for a mobile distributed system on Android platform using JXTA technology. Since the JXTA technology is neither a platform nor programming language dependent, I found that JXTA is an appropriate technology for P2P instant messaging and file sharing applications.

It was discovered in the project that it is possible for multiple users to send instant messages and share the resources in a P2P manner without depending on the central server. However, the size of a message or file to be sent must be 64 KB or less since the JxtaBiDiPipe used as a communication channel does not provide message chunking.

One point to keep in mind is that it’s not possible for rendezvous peer to differentiate multiple peers with the same login name. In such a case no peer could be able to get back a peer discovery response from the rendezvous peer. There fore peers in the JXTA network must have different login names to allow peers to discover each other.

I studied the JXTA Content Management Services (JXTA-CMS) which allow users of JXSE to search discover and download the resources within the JXTA network. Since new smart phones devices offer advanced capabilities the same as computers, users can for example create and delete directories, add, delete or modify files, add the access permissions to files in directories. Therefore, a P2P applications developer on a JXTA platform who is interested in working with the Android platform should look for a possibility to develop P2P applications which use CMS features thus users will be able to search for and download resources within the JXTA network rather than wait for the owner of the resources to send them.

## REFERENCES


1 Android Platform [online] 26 September 2009,

URL: http://developer.android.com/guide/basics/what-is-android.html Accessed 1 march 2010.

2 Android Devices [online] 20 February 2010,URL:http://androidcommunity.com/18-android-devices-on-market-by-end-of-2009-says-google-20090528/ Accessed 1 march 2010

3 Joseph D Gradecki, Mastering JXTA: Building Java P2P Applications, USA: John Wiley&Sons: Wiley Publishing Inc; 2002.

4 Brendon J Wilson, JXTA, United States of America: New Riders Publishing; 2002.

5 Sun Microsystems Inc, JXTA Java Standard Edition v2.5 Programmers Guide, United States of America: Sun Microsystems; 2007.

6 JXTA for Java ME [online] 25 September 2009,URL: http://developers.sun.com/mobility/midp/articles/jxme/ Accessed 3 march 2010.

7 Project PeerDroid [online] 25 October 2009 , URL: http://code.google.com/p/peerdroid/ Accessed 2 March 2010.

8 Android Chat [online] 12 October 2009, URL: http://code.google.com/p/androidchat Accessed 2 March 2010.

9 Daniel Brookshier, Darren Govoni, NavaneethKrishnan, Juan Carlos Soto, JXTA: Java P2P Programming, United States of America: SAMS Publishing; 2002.

10 AndExplorer [online] 2 March 2010, URL:http://www.lysesoft.com/products/andexplorer/index.html Accessed 6 march 2010

11 IMS SIP [online] 12 October 2009, URL: https://labs.ericsson.com/apis/mobile-java-communication-framework/blog/develop-ims-sip-client-android-part-1 Accessed 6 March 2010

12 JXTA [online] 25 September2009, URL:http://tim.oreilly.com/pub/a/p2p/2001/04/25/jxta.html Accessed 6 March 2010

13 Client-server architecture [online] 27 September 2009, URL: http://www.javaworld.com/javaworld/jw-10-2001/jw-1019-jxta.html Accessed 15 April 2010

14 JXTA architecture [online] 20 September 2009, URL: http://www2002.org/CDROM/refereed/597/ Accessed 15 April 2010

15 JXTA proxyless [online] 15 April 2010, URL: http://weblogs.java.net/blog/2008/02/02/new-jxta-micro-edition-cldcmidp-20 Accessed 15 April 2010

16 Rendezvous [online] 15 April 2010, URL: http://www.developer.com/java/j2me/print.php/1464091 Accessed 15 April 2010

17 Netbeans IDE [online] 15 April 2010, URL: http://netbeans.org/ Accessed 15 April 2010

18 Project JXTA [online] 15 April 2010, URL:http://jxta.dev.java.net Accessed 15 April 2010

19 Java ME [online] 15 April 2010, URL:http://java.sun.com/javame Accessed 15 April 2010

## APPENDIX A: RENDEZVOUS PEER

```
package metropolia.cap04;

import java.io.File;
import java.io.IOException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.logging.Level;
import java.util.logging.Logger;

import net.jxta.exception.PeerGroupException;
import net.jxta.id.IDFactory;
import net.jxta.peergroup.PeerGroup;
import net.jxta.peergroup.PeerGroupID;
import net.jxta.platform.NetworkConfigurator;
import net.jxta.platform.NetworkManager;
import net.jxta.rendezvous.RendezVousService;
import net.jxta.rendezvous.RendezvousEvent;
import net.jxta.rendezvous.RendezvousListener;



/**
* Responsible for the connection to the rendezvous peer.
*/
public class RendezPeer implements RendezvousListener {
	private boolean connected;
	private String Rend = new String("rend");
	private RendezVousService peerGroupRendezvous;

/**
* Constructor for rendezvous peer
*
* @param peerGroup
*/
	public RendezvPeer(PeerGroup peerGroup) {
		peerGroupRendezvous = peerGroup.getRendezVousService();
		peerGroupRendezvous.addListener(this);
	}

//start JXTA network
	private void start() {
```

```
  try {
    configureJXTA();
    System.out.println("executing JXTA");
    startJXTA();
    System.out.println("startJXTA(); done");
    createPeerGroup();
    System.out.println("waitForQuit(); started");
    waitForQuit();
  }
  catch (PeerGroupException e) {
    e.printStackTrace();
    System.out.println("Exiting.");
    System.exit(0);
  }
  catch (Exception e) {
        System.out.println("Unable to start JXTA platform.  Exiting.");
    e.printStackTrace();
    System.exit(0);
  }
}
      @Override

      public synchronized void rendezvousEvent(final RendezvousEvent event) {
   PeerID rdvPeerID = (PeerID) event.getPeerID();
   switch (event.getType()) {
      case RendezvousEvent.RDVCONNECT: {
         if (!view.contains(rdvPeerID)) {
            view.add(rdvPeerID);
         }
         break;
      }
      case RendezvousEvent.RDVRECONNECT: {
         if (!view.contains(rdvPeerID)) {
            view.add(rdvPeerID);
         }
         break;
      }
      case RendezvousEvent.RDVDISCONNECT: {
         if (view.contains(rdvPeerID)) {
            view.remove(rdvPeerID);
         }
         break;
      }
      case RendezvousEvent.BECAMERDV: {
         if (LOG.isInfoEnabled()) {
            LOG.info("Node became rendezvous (NAME: "
```

```
                    + group.getPeerGroupName() + ")");
            }
        }
        default: {
            break;
        }
    }



/**
* Get connected to a rendezvous peer.
*/
        public void waitForRdv() {
                System.out.println("Establishing connection to  rendezvous peer...");
                synchronized (Rend) {

                        while (!peerGroupRendezvous.isConnectedToRendezVous()) {
                                try {
                                                if
(!peerGroupRendezvous.isConnectedToRendezVous()) {
                                                        Rend.wait();
                                                }
                                        } catch (InterruptedException e) {

                                }
                        }
                }
                System.out.println("Connected to rendezvous peer");
        }

        public void setConnected(boolean connected) {
                this.connected = connected;
        }

        public boolean isConnected() {
                return connected;
        }
        //main method
        public static void main(String[] arg) throws PeerGroupException, IOException,
NoSuchAlgorithmException {
        Logger myLogger = Logger.getLogger("net.jxta");
        myLogger.setLevel(Level.ALL);
        RendezPeer myRend = new RendezPeer();
        myRend.connect();
                myRend.waitForQuit();
```

**APPENDIX B: METHODS FOR SENDING AND RECEIVING MESSAGES**

```
/*
         *
         *
         * send a series of messages over a JxtaBiDiPipe
         */
        public int sendMessage(JxtaBiDiPipe bidipipe) throws IOException {
                Log.d("P2PJXTAonANDROID"," sending the text messages..");
                long start = System.currentTimeMillis();

                Message msg = new Message();
                        try{
                                MessageElement sender = new
StringMessageElement("sender",peerName,null);
                                MessageElement contentElement = new
StringMessageElement("content",outMsg,null);

                                msg.addMessageElement(sender);
                                msg.addMessageElement(contentElement);
                                bidipipe.sendMessage(msg);
                        }catch(IOException ex){
                                ex.printStackTrace();
                        }
                        return Log.d("P2PJXTAonANDROID","Message sent");
        }

//Method for receiving messages

        public void pipeMsgEvent(PipeMsgEvent ev) {
                // TODO Auto-generated method stub
                final Message msg ;

                try{
                        msg = ev.getMessage();
                        if(msg == null){
                                return;
                        }
                        P2PJXTAonANDROID.handler.post(new Runnable() {
                                public void run() {
                                         msg.getMessageElement(P2PJXTAonANDROID.
                                        conversation.getText()+"\n"
+getCurrentTime()+msg);
```

```
                        }
                });
                Log.d("P2PJXTAonANDROID", "received message read");
                //calling the vibrator method
                p2pAnd.notifyMessageArrival();
            }catch(Exception e){
                e.printStackTrace();
            }

        }
}
```

## APPENDIX C: METHODS FOR SENDING AND RECEIVING FILE

```
//sending file data method
public boolean fileTransfer(String startDir) {

		int length = 0 ;
		byte[] buffer = new byte[300*1024];

		try {
			File file = new File("/sdcard/myphotos");
			FileInputStream in = new FileInputStream(file);

				while(length != -1){

				length = in.read(buffer);

				Message data = new Message();

				data.addMessageElement(new
StringMessageElement("Sender",
							peerName, null));
				//data.addMessageElement(new
StringMessageElement("fileData",fileData,null));
				data.addMessageElement(new
StringMessageElement("Filename", file
							.getName(), null));
				data.addMessageElement(new StringMessageElement(
							"FileSize", String.valueOf((Math.round((file
									.length() / 300*1024)) + 2)),
null));


				if (length != -1)
					data.addMessageElement(new
ByteArrayMessageElement(
								"data", null, buffer.clone(), null));
				else
					data.addMessageElement(new
ByteArrayMessageElement(
								"data", null, new byte[0], null));
							bidiPipe.sendMessage(data);
				}
			in.close();

		} catch (FileNotFoundException e) {
			// TODO Auto-generated catch block
```

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


}

//Receiving file data method

        public void recvFile( Message data) {
                sender = data.getMessageElement("sender").toString();
                filename = data.getMessageElement("Filename").toString();
                int ImageSize = Integer.valueOf(

        data.getMessageElement("ImageSize").toString()).intValue();
                int fileSize = Integer.valueOf(
                                data.getMessageElement("fileSize").toString()).intValue();

                final byte[] filedata = data.getMessageElement("filedata").getBytes(true);
                RandomAccessFile rand;
                try {
                    rand = new RandomAccessFile( FILE_FOLDER+"new
file"+filename,"rw");
                    if (packageSize != -1) {

                        rand.write(content, 0, fileSize);
                    } else {

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

                P2PJXTAonANDROID.handler.post(new Runnable() {
                    private MyBuddy myBuddy;
                        public void run() {
                        //MyBuddy myBuddy ;
```

```
        P2PJXTAonANDROID.conversation.setText(P2PJXTAonANDROID.
                                                conversation.getText() +"\n"
+getCurrentTime() + filename);
                        }
                });
                p2pAnd.notifyMessageArrival();
        }
```