Peer-to-Peer Programming with Wireless Devices

By

Tore Mørkved

Master Thesis in Information and Communication Technology

University of New South Wales, Sydney Australia

&

Agder University College, Grimstad Norway

Sydney, June, 2005

Abstract

Peer-to-Peer programming (P2P) has in recent years become a widely explored research area. With the evolution of wireless technology such as mobile phones, the idea to bring these two technologies together gives a new dimension to P2P communication, collaboration and resource sharing.

This master thesis explores the domain of Mobile Peer-to-Peer networking and proposes a Peer-to-Peer System with Wireless Devices. The system is based on an open, protocol-based P2P platform called JXTA. It allows any connected device on the network ranging from sensors and cell phones to personal computers and servers to communicate and collaborate in a Peer-to-Peer manner. It is platform and network independent and designed to be implemented on any networking device.

JXTA for J2ME (JXME) is a lightweight version of JXTA that gives P2P functionality to constrained wireless devices. The technology, which is open source, is under development by the JXTA community, and this thesis focuses on the development of JXME for the Connected Limited Device Configuration (CLDC).

The system proposed uses the JXME API, but suggests a more specific approach to implement different Peer operations such as Peer discovery, resource advertising and file transfer. Because of the limitations of wireless devices, one or more powerful Peers need to participate in the network as Proxy Services. This gives both the advantages of a fixed P2P network and the mobility of a wireless device.

The prototype developed demonstrates the P2P system with simple collaboration and file sharing. The application has been successfully tested on phone emulators, and network tests show that the system works in a controlled environment. Large file transfer is stable, but highly limited by memory constraints of the device.

Preface

This report contains the documentation of the master thesis “Peer-to-Peer Programming with Wireless Devices” which was conducted during 20 weeks from February to June 2005. The thesis was written in Sydney, Australia for the University of New South Wales (UNSW) for Agder University College, Grimstad, Norway.

My supervisor at UNSW has been Mr. Nandan Paramesh Parameswaran, and my contact at Agder University College has been Mr. Magne Arild Haglund.

I would like to thank my girlfriend, Bella, for her support during the writing process.

Tore Mørkved

Sydney, June 2005

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Peer-to-Peer programming (P2P) has in recent years become a widely explored research area and gets a lot of attention in both media and the computer industry. It has changed the way we communicate, collaborate and share resources. Implementations such as MSN Messenger or Skype have become almost mandatory for internet users, enabling instant communication in a way that makes E-mail appear antiquated. Internet traffic is now mainly dominated by P2P networks for file sharing, such as Gnutella and BitTorrent [6].

Personal wireless devices such as mobile phones have evolved from pure cell phones to small all-in-one devices that, in addition to being phones, contain camera, radio, mp3 player and the ability to connect to the Internet. The computing resources are constantly increasing, unlocking a great potential; to have personal wireless devices participate as Peers on a Peer-to-Peer network!

The obvious possibilities of instant messaging and file sharing can very likely become as popular as they have on a traditional P2P network, but an additional property will lead to new possibilities; the mobile phone is a *personal* device. It is always with us, it defines us and represents us at all times, wherever, whenever. This can be exploited in many ways, for instance in collaboration over large distances to solve a common problem, or personal monitoring in medical situations.

1.2 Problem specification

P2P programming paradigm is increasingly becoming a dominant mode of resource sharing and cooperative problem solving. Traditionally, a peer is a computing device with substantial amount of computing power and resources. However, it will be an interesting idea if small wireless devices (such as mobile phones and wireless sensors) are also made peers. In this context, programming wireless devices has many challenges. The primary objective of this thesis work is to build a P2P system consisting of wireless devices only.

In particular, we plan to investigate the following problems:

1. How do the limitations of the wireless devices (such as total power remaining in the battery, losing communication with other peers at anytime, small displays, and

limited CPU power and memory) affect the solutions to a problem, and how do they affect the program logic?

2. In dynamic emergency situations, how is programming or computing distributed amongst the other wireless peers so that in case of power failure or disruption in communications, the network can ensure a graceful degradation?

We propose to create a mobile P2P scenario and build a prototype in Java based on the system design using phone emulators.

1.3 Objectives and Limitations

1.3.1 Objectives

The main objective of this thesis is to build a Peer-to-Peer system for wireless devices. The system design will be based on an open, protocol-based P2P platform called JXTA, which enables the developer to focus on the end-system instead of the extensive task of creating the P2P network. The prototype will be developed in Java for the Java 2 Micro Edition (J2ME) platform.

Firstly, J2ME, P2P networking and the JXTA technology will be investigated. Based on this a Mobile Peer-to-Peer system will be designed and simple a prototype will be implemented using phone emulators such as Sun’s Wireless Toolkit.

Next, the application will be tested to see how it performs in a scalable network and how it performs when transferring large files.

1.3.2 Limitations

The p2p system will not be tested in a large scale due to resource and time constraints. Non-functional testing, such as stability, security and usability are not prioritised, but should be explained and considered if time and resources allow it.

1.4 Reader's Guide

This reader's guide is included to give an overview of the report by summarising the chapters and describing how they relate to one other. It gives an overview of the report, making it easy to find the parts the reader is interested in without reading through the whole report.

Chapter 1 Introduction: This chapter is an introduction to the report. It contains motivation, problem specification, objectives and limitations, in addition to this reader's guide.

Chapter 2 Literature Study: Review of background information I have researched before designing the P2P system. The main topics are Mobile devices, J2ME, P2P, JXTA and JXME.

Chapter 3 System Design: Contains the design of a proposed P2P system for mobile phones. The foundation for this chapter is Chapter 2.

Chapter 4 Prototype Design: Contains the design of a prototype application. It is based on Chapter 3, the System Design.

Chapter 5 Performance Results: This chapter presents the results of the prototype designed in chapter 4, with test documents for functionality test, network test and large file transfer test.

Chapter 6 Discussions: Analyzes and discusses my contribution to this thesis (chapter 3 4 and 5) ,in regards to the problem specification in chapter 1 and literature study in chapter 2.

Chapter 7 Conclusions: The conclusions for this thesis. It continues and concludes the discussions in chapter 6. Also, it proposes future work on the domain.

2 Literature Study

2.1 Introduction

This chapter will summarise topics that have been explored as a foundation for my own contribution to this thesis. References to relevant sources with domain elaboration are referred to and found in the References chapter.

I have investigated today's research and development in the area of Mobile devices, Java 2 Micro Edition (J2ME), P2P technology and the JXTA framework. Some topics have been explained in more detail for the reader to better understand the remainder of the thesis.

2.2 Mobile devices

2.2.1 Introduction

The Mobile phone has become a necessity in our lives; it is always with us, it is always on. It has become so much more than just a phone; it has become an extension of who we are. As the mobile phone development continues its astonishing progress, we are now experiencing a fusion of technology into a mobile device, not only being able to make phone calls, but also including radio, music player, mega pixel camera, camcorder and internet connection to name a few properties.

In this chapter I will give a short overview of today's mobile phones' constraints and limitations.

2.2.2 Constraints

A mobile phone still has many constraints compared to larger computing devices, such as personal computers. In the Table below, I have listed a few mobile devices with relevant constraints and possibilities. Even though more powerful phones exist, this table represents what is considered modern phones at the time of writing (June 2005). Interesting properties are:

- Screen resolution, size and colour
- Data transfer type
- Memory size, Heap memory (RAM)
- Memory size, persistent memory to store files such as images or music.
- Maximum JAR size, determines how large an application can be.
- Supported APIs

The information is extracted from [33] and all the images of the phones are taken from this website.

Table 1 – Mobile phones, specifications

	Name	**Nokia 6110**	Heap size	512 kB
	Screen size	128x160	Max JAR size	128 kB
	Colours	65 000	Memory size	3,5 MB
	Data	EDGE	Supported API	CLDC 1.1, MIDP 2.0
	Name	**Nokia 6680**	Heap size	Dynamic
	Screen size	176x208	Max JAR size	Dynamic
	Colours	262 000	Memory size	10 MB
	Data	3G Data	Supported API	CLDC 1.1, MIDP 2.0
	Name	Motorola V220	Heap size	800 kB
	Screen size	128x128	Max JAR size	100 kB
	Colours	65 000	Memory size	2 MB
	Data	GPRS	Supported API	CLDC 1.0, MIDP 2.0
	Name	**Sony Ericsson K700**	Heap size	1.5 MB
	Screen size	176x220	Max JAR size	300 kB
	Colours	65 000	Memory size	32 MB
	Data	GPRS Class 10 (48 kbps)	Supported API	CLDC 1.1, MIDP 2.0
	Name	**Nokia 6630**	Heap size	Dynamic
	Screen size	176x208	Max JAR size	Dynamic
	Colours	65 k	Memory size	10 MB + Memory Card
	Data	3G Data	Supported API	CLDC 1.1, MIDP 2.0
	Name	**Nokia 6230i**	Heap size	512 kB
	Screen size	128.128	Max JAR size	128 kB
	Colours	65 000	Memory size	3,5 MB + memory card
	Data	EDGE	Supported API	CLDC 1.1, MIDP 2.0

2.3 Java 2, Micro Edition (J2ME)

2.3.1 Introduction

This chapter gives an introduction to the Java 2 Micro Edition platform, with focus on its capabilities as a Peer-to-Peer application platform, its possibilities and its limitations. I will examine the Connected Limited Device Configuration (CLDC) and the Mobile Information Device Profile 2.0 (MIDP 2.0), which is the newest addition to the J2ME platform.

The Java programming language was originally developed for consumer electronic devices, but over the years it evolved into a set of technologies used primarily to develop desktop and server-based applications.

So, in a way, you can say that the latest contribution to Sun's Java Platform, the Java 2 Micro Edition (J2ME) is returning to the very origins of Java technology. J2ME is descending from several early java platforms for small devices: The Oak part of the Green project (early 1990's), Java Card (1996), Personal Java (1997), EmbeddedJava (1998) and the Spotless System and the K virtual machine (1999). The Java Card technology is still separated from the J2ME platform, and remains an important technology based on smart cards [14].

2.3.2 Overview of the Java 2 Platform

Java 2 Micro Edition is a subset of the Java 2 Platform. It is a platform for small embedded devices and consumer devices, unlike the other two Java 2 platforms; J2SE (Standard edition) *"provides a runtime environment and a complete set of APIs for desktop applications, and defines a core set of functionality for the other editions" [14].*The third platform is the J2EE (Enterprise edition), which is a superset of the J2SE. It supports *"scalable, transaction-oriented and database-centred enterprise programming"* [14].

Unlike J2SE, J2ME is not a piece of software, nor is it a single specification. J2ME is a collection of technologies and specifications that are designed for different parts of the device market. To be able to provide a specialised solution to different devices, J2ME is divided into *configurations, profiles,* and *optional packages.*

Figure 1 shows the J2ME platform Layer stack with the J2EE, J2SE and Java Card Layer stack.

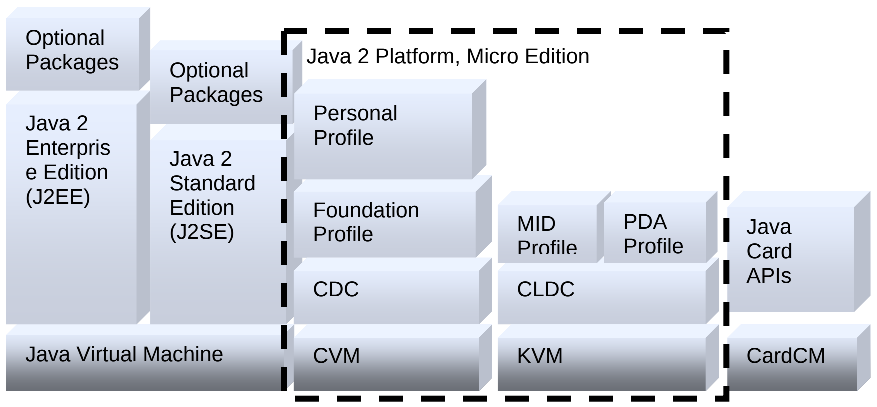


Figure 1 - Java 2 Micro Edition Platform

J2ME has a software layer stack consisting of three layers on top of the Operating System of the device; the Java Virtual Machine (JVM) Layer, Configuration Layer and Profile Layer. However, before we go deeper into the different layers of the J2ME, it is important to understand where J2ME is standing with respect to other Java technologies for embedded devices.

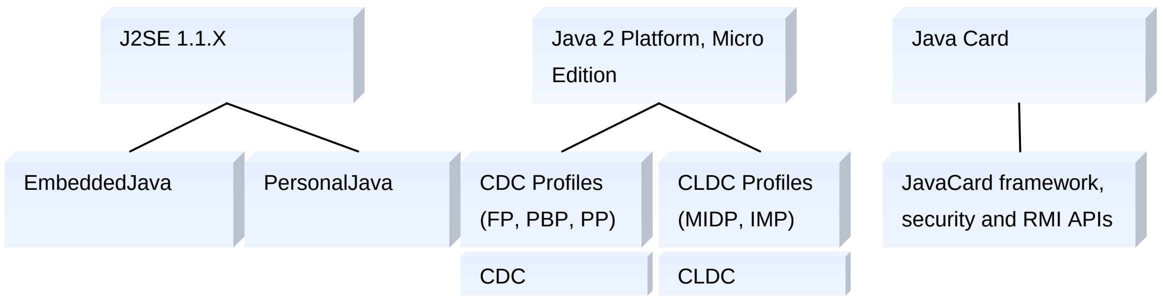

J2SE 1.1.X
Java 2 Platform, Micro Edition
Java Card
EmbeddedJava
PersonalJava
CDC Profiles (FP, PBP, PP)
CLDC Profiles (MIDP, IMP)
JavaCard framework, security and RMI APIs
CDC
CLDC

Figure 2 - Variants of the Java Platform for Embedded Devices

As figure 2 demonstrates, Java offers EmbeddedJava, PersonalJava, J2ME and Java Card. EmbeddedJava and PersonalJava has gone through the Sun “End of Life” (EOL) process and all developers are encouraged to move to the J2ME family of products, since these technologies are being migrated into J2ME, and are no longer supported by Sun Microsystems [15], [16]. Finally, Java Card technology supports development on Java based smart cards [17].

2.3.3 Configuration Layer

A configuration defines a basic, lowest-level J2ME runtime environment. This includes the virtual machine and a set of core classes derived primarily from J2SE. As seen in figure 4 and 5, J2ME has been divided into two configurations, the Connected Device Configuration (CDC) [18], [19], [20] and the Connected Limited Device Configuration (CLDC) [21], [22], [23].

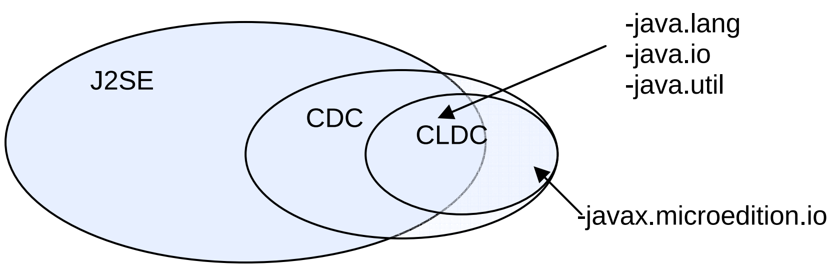


Figure 3 - Relationship between J2SE and the J2ME configurations

The Connected Device Configuration (CDC)

The Connected Device Configuration (CDC) is designed for more powerful devices, such as high-end cell phones and PDAs and the more sophisticated devices such as set-up boxes, or car navigation systems.

These devices have 32-bit processors, at least 2MB of main memory, 2.5 MB of ROM, and some type of network connectivity. The CDC uses a Java Virtual Machine with full J2SE capabilities.
As figure 3 shows, the CDC is a superset of CLDC; it includes the CLDC classes, including those not included in J2SE, and has additional classes that CLDC does not have.

The Connected Limited Device Configuration (CLDC)

CLDC is a minimal J2ME configuration for devices with substantial constraints on computing power, battery life, memory and bandwidth. The CLDC 1.1 specification [22] assumes these technical requirements and characteristics for a device:

- At least 160 kb of total memory available for the Java platform
- Processor speed starting from 8 – 32 MHz.
- 16/32 bit processor.
- Limited power, usually battery operation.
- Connectivity to some type of network, although with possibly limited (9600 bps or less) bit rate.
- High-volume manufacturing (possible millions of units)
- User interfaces with varying degrees of sophistication down to and including none.

The Specification addresses the scope of CLDC1.1 to include:

- Java language and virtual machine features

- Core Java libraries (java.lang.*, java.util.*)
- Input/Output (java.io.*)
- Security
- Networking: General framework for network connection
- Internationalization: Handles different character encodings

Also, it specifies the CLDC1.1 to *not* include:

- Application installation and life-cycle management
- User interface functionality
- Event handling
- High-level application model (the interaction between the user and the application

These features are to be addressed by the profiles implemented on top of the CLDC; the most common used is the Mobile Information Device Profile (MIDP), which will be described in the next subchapter.

2.3.4 Profile Layer

The Profile Layer defines the minimum set of APIs available on a device family, for example a mobile phone or a PDA. The profile usually includes more domain-specific libraries than what is included in the configuration. As a result, applications are written for a particular profile and implicit use the configuration the profile is built upon. Different profiles exist for different types of devices. The two profiles existing for the CLDC are the Information Module Profile (IMP) and Mobile Information Device Profile (MIDP).

Mobile Information Device Profile (MIDP) [27] was the first J2ME profile, and is the most mature and widely used, with millions of deployments around the world, mainly on PDA's and on cell phones and other handheld communicators. The new MIDP 2.0 [30] has enhanced the profile's capabilities concerning new networking (TCP Sockets and UDP datagrams, secure connections), and also a robust security API to support TCP socket streams, and even API's for gaming.

Information Module Profile (IMP) [28] targets devices with little or no capabilities for user interface, such as headless embedded devices in vending machines. The new IMP-NG (Next Generation) will take advantage of MIDP 2.0's new security and networking types and APIs.

The table below summarizes the packages available in MIDP and IMP 1.0 [14].

Table 2 - Packages in the CLDC - Based Profiles

Name	Description	MIDP 1.0	MIDP 2.0	IMP 1.0
Java.lang	MIDP subset of the core Java programming language	X	X	X
Java.util	Small subset of utility classes	X	X	X
Java.io	MIDP subset of system input and output through data streams	X	X	X
javax.microedition.io	Networking support using the Generic Connection Framework; includes new socket, UDP, serial, and secure connection types, and push functionality	X	X	X
javax.microedition.lcdui	MIDP classes for user interface	X	X	
javax.microedition.lcdui.game	Gaming classes such as sprites, game canvas, and layer manager		X	
javax.microedition.media	Interfaces for controlling (Control) and rendering (Player) audio–sound classes compatible with the Mobile Media API specification [37]		X	
javax.microedition.media.control	Sound-control classes (ToneControl and VolumeControl) - compatible with the Mobile Media API specification [37]		X	
javax.microedition.midlet	The application interface, its life-cycle classes, and its interactions with the runtime environment and the application manager	X	X	X
javax.microedition.pki	Public key class for certificates used to authenticate information for secure connections		X	X
javax.microedition.rms	Classes for storing and retrieving persistent data	X	X	X

2.3.5 MIDlet Memory

Memory is always a resource constraint when we develop applications on handheld devices. There are in fact three memory types on a MIDP device; program memory, heap memory and persistent storage [32].

Program Memory

This memory is available to store a MIDlet on the device. Some device manufacturers limit the allowed size of each MIDlet. This changes from device to device. Table 1 shows some examples of program memory limits.

Heap Memory

This is the runtime memory, or the RAM of the device. When the application is running, all local variables and member variables are allocated from the heap memory. On a J2SE device, the heap can constitute hundreds of megabytes, but on J2ME devices it is very limited. Table 1 lists heap size on several mobile devices.

Persistent Storage

This memory is used for record stores in MIDP called Record Management System (RMS). If a File Connection API [35] is available, the application can use resources on the phone’s file system and memory cards.

2.3.6 Summary

J2ME has, with the CLDC and MIDP 2.0 specifications, set a standard for mobile phones to follow. This chapter has listed some of the most important possibilities and limitations of mobile devices implementing J2ME.

2.4 Peer-to-Peer (P2P) Networking

2.4.1 Introduction

This chapter defines and introduces the domain of Peer-to-Peer (P2P) networking. It gives a short overview of the evolution of different P2P systems and how resource discovery can be implemented. At the end of this chapter, it summarises some advantages and disadvantages of P2P compared to traditional client/server architecture.

2.4.2 What is P2P?

Wikipedia, the free encyclopaedia [31] defines Peer-to-Peer as:
“A peer-to-peer (or P2P) computer network is a network that relies on computing power at the edges (ends) of a connection rather than in the network itself.”
(…)
“A pure peer-to-peer file transfer network does not have the notion of clients or servers, but only equal peer nodes that simultaneously function as both "clients" and "servers" to the other nodes on the network.”

The P2P concept is currently sweeping through both the computing industry and the media, and the implementation is commonly seen in instant messaging (IM) applications such as ICQ or MSN Messenger, and different file sharing applications, such as Kazaa or Gnutella. Conceptually, P2P is much more – or much less – than that; P2P can simply be two or more PCs that are connected and share resources without going through a separate server. At the other extreme, one could say that the entire Internet operates very similar to a giant P2P network. From a broad perspective, the whole Internet it self consists of networked computers containing a wide selection of geographically separated data, communicating directly with one other. [4]

P2P could be explained by answering the question: “*How can you connect a set of devices in such a way that they can share information, resources, and services?*” [12]

It seems like an easy question, but if we dig deeper, we learn that this basic question derives more complex challenges:
How does one device learn from another device's presence, that is, how do we deal with *discovery*?

- How do devices organize *groups* of common interest?
- How does a device *advertise* its resources?
- How do we uniquely *identify* a device?
- How do devices *exchange data*?

Many P2P solutions have been created to provide answers to these questions. The problem is that they all have their own answer hard-coded into the implementation, giving no room for flexibility and interoperability. To evolve P2P into a mature solution platform, developers need to agree on a solid, well-defined base language to communicate and perform the fundamentals of P2P networking.

In the next chapter, I will have a look at one solution to this problem, a project called JXTA. But first, different network architectures will be presented; from the traditional client/server architecture to the ever evolving P2P architectures.

2.4.3 Client/Server Architecture

In the traditional client/server architecture the client sends a request to the server, which handles most of the processing involved in delivering the requested service, leaving the clients relatively unburdened.

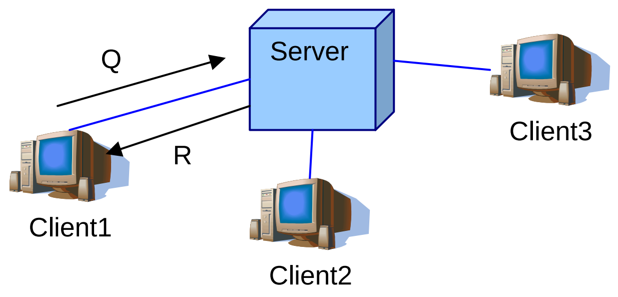


Figure 4 - Client/Server Architecture

Client:

- Sends query request (Q)
- Receive response (R)

Server:

- Receive query request (Q)
- Processes service requests
- Sends the result as a response to the client (R)

This architecture has major drawbacks. As the number of clients increase, the load on the server also increases, until the bandwidth or the processing power reaches its limits, preventing the server from handling additional clients. The advantage however, is that the client is left with very little responsibility, thus it does not require high computing power. Ultimately, this means that almost any device with a network connection can act as a client and receive server data.

2.4.4 P2P Network Architectures

There are mainly three network models of P2P, namely Centralized, Decentralized and Hybrid models.

Centralized Architecture

The first generation Peer-to-Peer system was initiated by the launch of Napster in May 1999. When this infamous application was at its peak in February 2001, it had 29.4 million registered users who shared 2.79 billion files in the same month [6]. Napster was based on a centralized index, which ultimately led to its downfall in late 2001, when it was forced by the record industry to shutdown.

Centralized network architecture uses a centralized indexed server to maintain a database of all the content and users at any time. The database is updated whenever a peer logs on to the network.

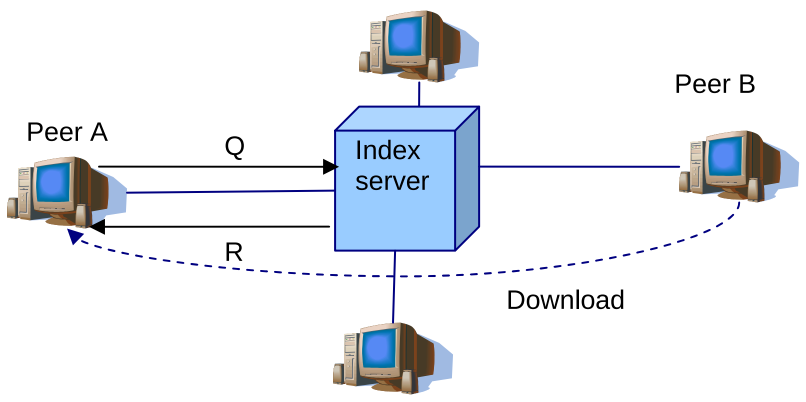


Figure 5 - Centralized Network Architecture

Peer A sends a query request to its index server. The index server uses the search request to query the database. If matches are found, the server returns the result to node A, telling him which node has the file, in this example, Peer B. Node A uses this information to start download from Node B.

Evaluation

This architecture allows a fast search response time, and is easy to implement and maintain. It provides a high degree of performance and resilience, but has a single point of failure. Because of this, it is vulnerable to censorship and technical failure. Popular data may become less accessible because of the load of the requests on a central server. Another disadvantage is that its central index might be obsolete, because the database is only refreshed periodically.

Decentralized Architecture

Second generation P2P uses a decentralized, distributed architecture to avoid the centralized weakness, “single-point-of-failure”. Instead of central servers, each peer acts as an index server, searches and holds its own local resources, and as a router, relaying queries between peers. An example is the Gnutella network.

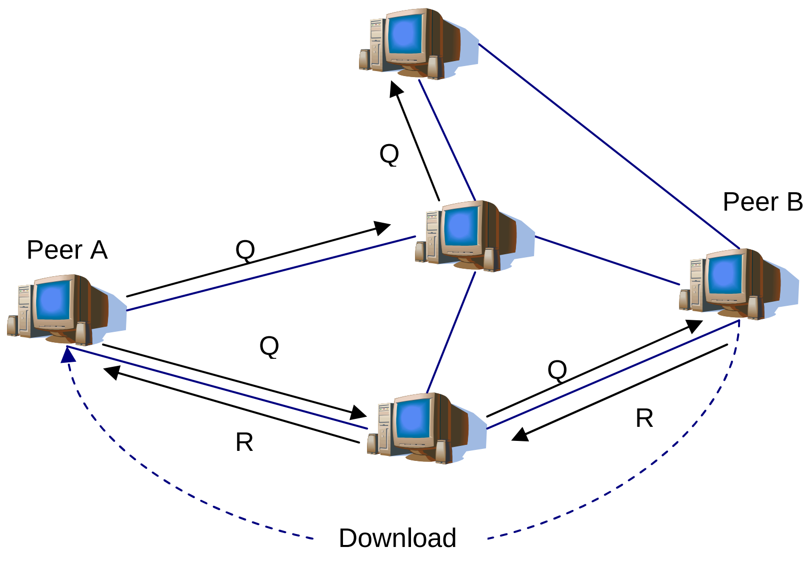


Figure 6 - Decentralized Architecture

Node A sends a query message to the peers it is directly connected to. These peers check their local list of resources to match the query and forward the query to the peers they are connected to on the network. This process continues, spreading the query across the network. If a peer, in this example Peer B, matches the query with its local resource, it returns a response message back across the network to Peer A. Peer A then downloads the resource directly from Peer B

Evaluation

Each peer is directly connected to a number of other peers. Relaying queries and result messages between peers generates large network traffic (chatter). It also results in slow information discovery compared to a centralized architecture. The system avoids single point of failure, which means it is resistant to crashing and shutdowns. It also scales inherently.

Hybrid Architecture

Third generation P2P is a hybrid of centralized and distributed, combining the best of both architectures. It deploys a hierarchical structure by establishing a backbone network of Super Nodes that take on the characteristics of a central index server. When a client logs on to the network, it makes a direct connection to a single Super Node which gathers and stores information about peer and content available for sharing. An example of a hybrid P2P network is the Direct Connect (DC) network.

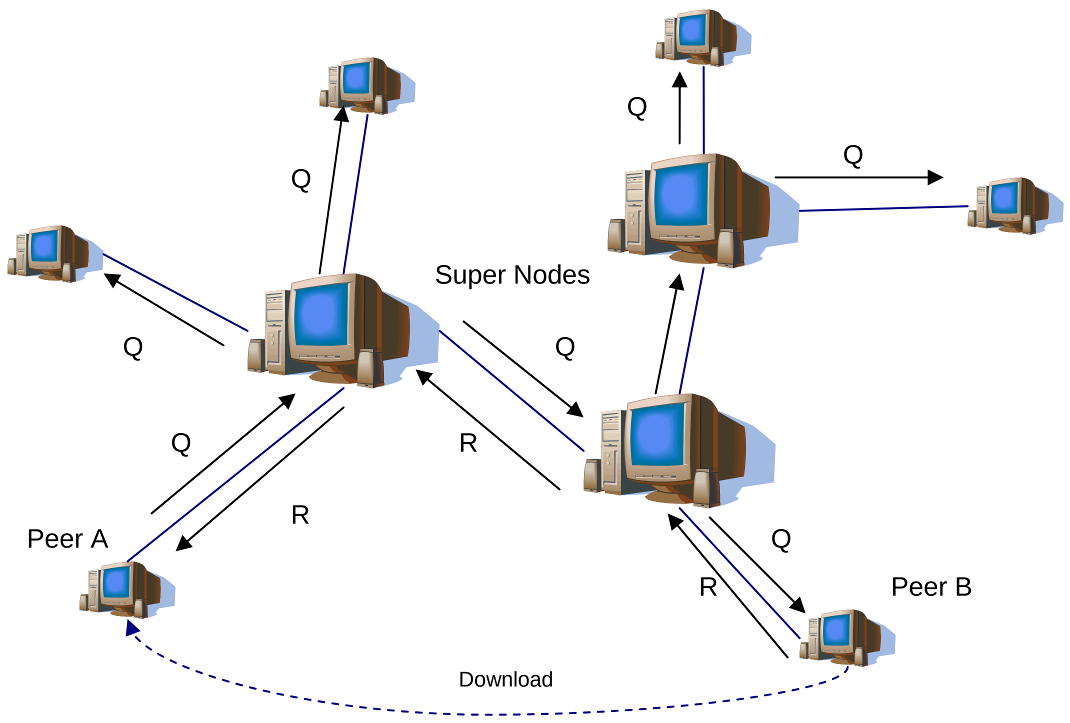


Figure 7 - Hybrid Architecture

Peer A sends a query message to its local Super Node. The Super Node runs the query in its own index and disseminates the query to other Super Nodes on the network. The query response are returned to Super Node which in turn relays the results to Peer A. Peer A then downloads the resource directly from Peer B.

Evaluation

The use of Super Nodes improves the search response times and generates less overhead traffic on the network than decentralized networks. The Super Nodes also reduce the workload on central servers in comparison with fully centralized indexing systems such as Napster. The single point of failure or control diminishes as the number of Super Nodes increases.

2.4.5 Resource Discovery

Discovering resources can be handled in several ways. Brendon Wilson [12] implies three main methods: No discovery, direct discovery and indirect discovery.

No discovery

Peers relay on a cache of previously discovered advertisements.
This reduces network traffic, but the information can become obsolete and increase network traffic by trying to discover a resource that no longer exists at given peer and then resort to active discovery.
To reduce the possibility of a given advertisement becoming obsolete, a cache can make advertisements expire, thereby removing them from the cache based on the probability that a given advertisement is still valid.

Direct Discovery

Peers that exist on the same LAN might be capable of discovering each other directly without relying on an intermediate rendezvous peer to aid the discovery process. Direct discovery requires peers to use the broadcast or multicasting capabilities of their native network transport.
Unfortunately, this discovery technique is limited to peers located on the same local LAN segment and usually can't be used to discover peers outside the local network. Discovering peers and advertisements outside the private network requires indirect discovery conducted via a rendezvous peer.

Indirect Discovery

Indirect discovery requires the use of a Super Peer to act as a source of known peers and advertisements, and to perform discovery on a peer's behalf.
This technique can be used by peers on a local LAN to find other peers without using broadcast or multicast capabilities, or by peers in a private internal network to find peers outside the internal network.

2.4.6 Why Peer-to-Peer networking

The potential of P2P reaches far beyond the recent years media focus on the area, namely through distribution of copyrighted material such as mp3's and movies.

The advantages and disadvantages of P2P are usually compared to the traditional client/server technology. Some advantages are:

***Distributed computing power*:** In his book, Brendon Wilson [12] presents this example to show the enormous amount of potential computing power and storage we have on client machines around the globe:

"Assume, with a lot of modesty, that 10 million 100 MHz machines are connected to the Internet at any time, each possessing only 100MB of unused storage space, 1000bps of unused bandwidth, and 10% unused processing power. At any time, these clients represent 10 petabytes (10^{15} bytes) of available storage space, 10 billion bps of available bandwidth, and 10^5 GHz of wasted processing power! P2P is the key to realizing this potential" [12].

No single point of failure: Removing the centralized server, which can be subject to crash, failure or overload would provide a more robust system which could withstand major disasters or other events that would result in downtime.

Distributed search: The Internet is a network of under utilized resources, partly due to the traditional client-server computing model. Take web searching for instance; no single search engine can locate and catalogue the ever-increasing amount of information on the Web at an acceptable speed [1]. Google claims that it searches over 8 billion web pages (June 2005), but this is just a small part of the World Wide Web. Consider all the private storage on client machines, and all the databases containing data the web engines have no access to. Using P2P technology and giving each Peer a responsibility to search its own domain would produce a much larger, more accurate and more updated search result.

2.5 Project JXTA

This chapter introduces the JXTA technology, gives a brief overview of JXTA architecture and defines concepts and terminology. It also looks at advantages and disadvantages of JXTA as a P2P platform.

2.5.1 Introduction

JXTA [3] is derived from the word Juxtapose, meaning side by side. “*It is a recognition that peer-to-peer is juxtaposed to client - server or Web based computing – what is considered today's traditional computing model*“[12].

JXTA technology is a set of open protocols that allows any connected device on the network ranging from cell phones and wireless PDAs to PCs and servers to communicate and collaborate in a P2P manner. JXTA technology is now distributed under an open-source license, and as such, is being co-developed by a larger community of users interested in P2P computing [9].

JXTA peers create a virtual network where any peer can interact with other peers and resources directly even when some of the peers and resources are behind firewalls and NATs or are on different network transports [3].

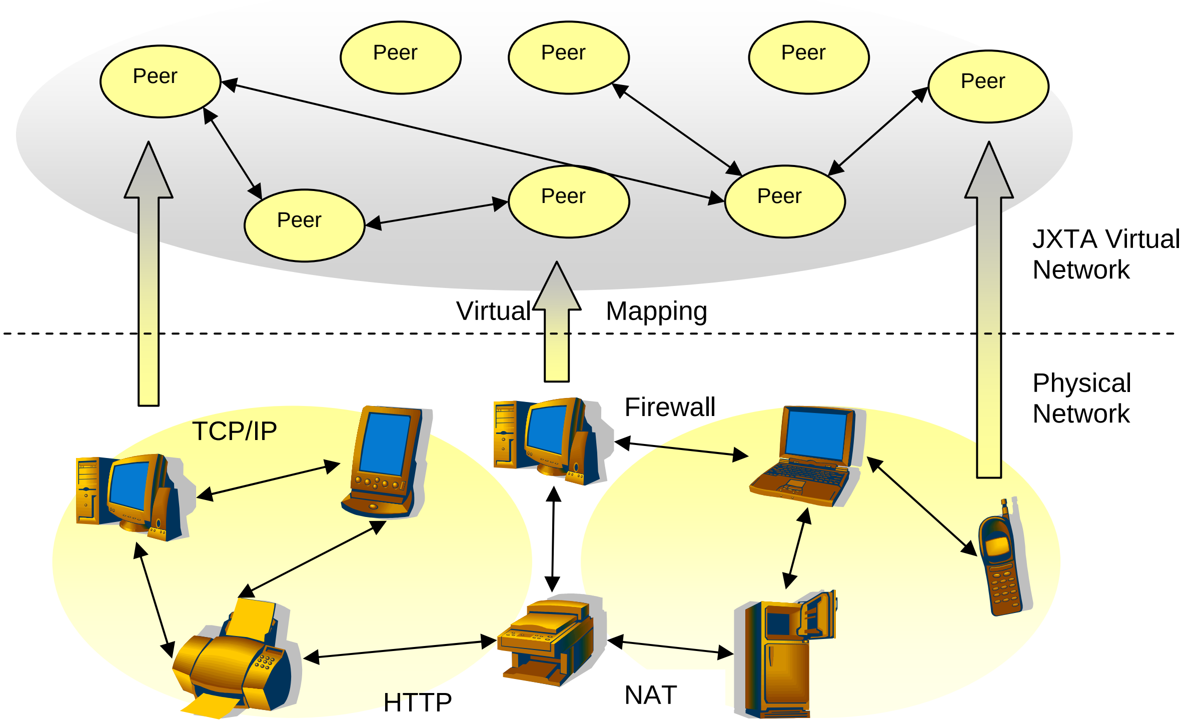


Figure 8 - Virtual mapping of a JXTA network

Project JXTA was conceived with a set of objectives intended to address the shortcomings of the peer-to-peer systems already in existence or under development, such as:

- ***Interoperability***. "*JXTA technology is designed to enable interconnected peers to easily locate each other, communicate with each other, participate in community-based activities, and offer services to each other seamlessly across different P2P systems and different communities"* [9]. *Wikipedia* [31] lists over 60 different P2P networks and all of these have their own communities, and few are able to operate together, even though they have the same purpose. If the underlying infrastructure and protocols had been standardised, all communities would have been able to communicate and collaborate.
- ***Platform independence***. *"JXTA technology is designed to be independent of programming languages (such as C or the Java programming language), system platforms (such as the Microsoft Windows and UNIX operating systems), and networking platforms (such as TCP/IP or Bluetooth)"* [9]. The majority of P2P solutions today assumes the use of TCP, and can not operate in any other environment. Flexible P2P solutions need a language that explicitly declares all of the variables in any P2P solution.
- ***Ubiquity***.*" JXTA is designed to be implemented on any device with a digital heartbeat, including sensors, consumer electronics, mobile phones, PDAs, appliances, network routers, desktop computers, servers, and storage systems."* [9] In doing so, the technology is set for the future vision of interconnecting all sorts of electronic devices.

2.5.2 JXTA Architecture

The JXTA architecture can be broken into three layers, *The Core, the Services* and *the Applications*, as seen in the figure below.

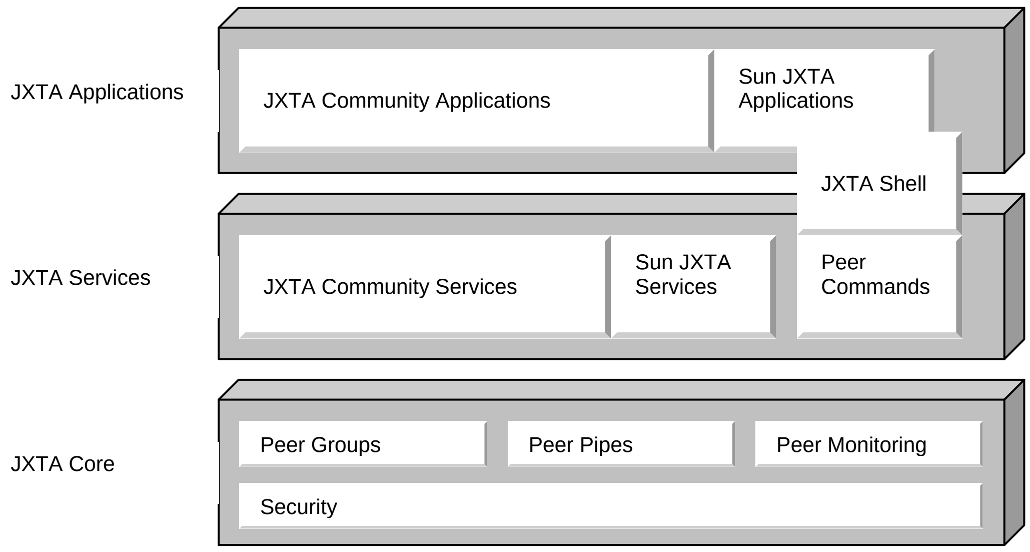


Figure 9 - The JXTA three-layer architecture

Each layer is built on the capabilities of the layer below.

The Core Layer

The core layer provides the essential elements of a P2P platform. These are the elements that ideally would be agreed upon and shared by all P2P solutions:

- Peers
- Peer Groups
- Network Transport (Pipes, Endpoints, Messages)
- Advertisements
- Entity Naming (Identifiers)
- Security and Authentication Primitives
- Protocols (communication, discovery, monitoring)

All these elements are described in the next chapter; the terminology. Note that JXTA’s six main protocols are implemented as services, but located at the Core layer to distinguish them from the service solutions in the Service layer. All the other aspects of a JXTA P2P solution are built on this core layer.

The Services Layer

The services layer provides optional P2P services, such as the following:

- Searching for resources at a Peer

- Sharing documents from a Peer
- Performing peer authentication

"Services are built on top of the JXTA platform to provide specific capabilities that are required by a variety of P2P applications and can be combined to form a complete P2P solution" [12]. This is also the layer where community-specific services can be developed to extend the services already given by JXTA.

The Applications Layer

This layer contains the common P2P applications we know, such as file sharing and instant messaging applications. The applications layer is built on the capabilities of the services layer, and provides a user interface for the user to invoke the services. The JXTA shell is shown as both an application and a service because it is a collection of services invoked as peer commands, providing only a minimal user interface.

2.5.3 Terminology

This is an introduction to the terminology and concepts of JXTA, and their relation to the general framework that is common to all P2P networks. JXTA uses a certain terminology that requires a familiarity with in order to understand the framework.

Advertisements

All network resources in JXTA, such as Peers, PeerGroups, Pipes and Services are represented by *advertisements*. Advertisements are *"language-neutral metadata structures resource descriptors represented as XML documents"* [13].

The figure below shows a PeerGroup Advertisement (jxta:PGA):

```
<?xml version="1.0"?>
<!DOCTYPE jxta:PGA>
<jxta:PGA xmlns:jxta="http://jxta.org">
<GID> urn:jxta:jxta-NetGroup</GID>
<MSID>urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE000000010206</MSID>
<Name>NetPeerGroup</Name>
<Desc>NetPeerGroup by default</Desc>
</jxta:PGA>
```

Figure 10 - Structure of a PeerGroup Advertisement

The element tag <GID> is the unique PeerGroup ID. <MSID> is the module specification ID which identifies the advertisement that describes all services available in this PeerGroup. <Name> is the name of the group, and <Desc> is the optional description of the group.

JXTA ID's

Every network resource (Peer, Pipe, data, PeerGroup etc.) is assigned a unique JXTA ID. These ID's are abstract objects enabling multiple ID representations (IPv4, IPv6 or MAC addresses) to coexist in the same JXTA network. It is using 128-bit random UUIDs allowing each peer to generate its own IDs.

A single peer supporting multiple network interfaces (Ethernet, Wi-Fi, etc.) will have the same Peer ID, even if they switch between networks.

Peers

Peers are nodes on a P2P network. A Peer is not limited to act as an application running on a computer connected to a network; the JXTA definition suggests that a peer might just as well be an application distributed over several machines, or that it might be a smaller device, such as a mobile phone, that connects indirectly to the network. A single machine or a single application might even be responsible for running multiple peer instances.

The JXTA book [12] defines a peer as follows:

"Any entity capable of performing some useful work and communicating the results of that work to another entity over a network, either directly or indirectly"

There are three possible types of peer in a P2P network:

Simple Peers

- Serving a single end user
- Typically hidden behind a firewall or Native Address Translation (NAT) equipment
- Have very little responsibility on a P2P network

Rendezvous Peers

- Rendezvous means "gathering" or "meeting place, implying that a rendezvous peer provides peers with a network location to discover other peers and resources.
- Forwards messages to all its known peers
- Can cache information to improve responsiveness and reduce network traffic.

Router Peers

- Enables other peers to communicate through a firewall or NAT.
- This Peer provides routing information to perform mapping between a unique ID specifying a remote peer and a representation that can be used to contact the peer via router peer.

PeerGroups

The JXTA book [12] defines a PeerGroup as:
"A set of peers formed to serve a common interest or goal dictated by the peers involved. PeerGroups can provide services to their member peers that aren't accessible by other peers in the P2P network."

The main purpose of peer groups is to subdivide the JXTA space into smaller, more private parts, which is necessary considering that all P2P applications would share the same protocols. For example, a messenger application might use one group, and a file sharing application, such as Gnutella, would use another group. The groups are divided based on the following [12] [13]:

Mutual interest: Peers who connect with the same goal, or are using the same application or service may want to form a group to keep the service private among the group members and to avoid unnecessary network traffic.

Security: A peer group can create restrictions by employing authentication services. It limits the access to the peer group and to its services.

Monitoring: Peer groups permit monitoring of peers for any purpose, for instance heart condition, status information or traffic introspection.

On boot time, every peer joins the NetPeerGroup, which is the root group that every peer belongs to initially. A peer group provides a set of core services specified by JXTA:

- Discovery Service
- Membership Service
- Access Service
- Pipe Service
- Resolver Service
- Monitoring Service

If these core services are inadequate for a more demanding PeerGroup, additional services can be developed for specific group purposes.

Pipes

The network transport layer handles the data transmission over the network, including breaking the data into manageable packets, adding appropriate headers and, in some cases, ensures packet arrival to its destination. The transport protocol is not fixed; it can be low-level, such as TCP or UDP, or high-level, such as HTTP or SMTP.

JXTA handles the communication through Pipes. Pipes are virtual communication channels used to send and receive messages. They create the illusion of virtual in and out pipes that a peer can send data on or listen to. Pipes can connect to one or more *endpoints* referred to as *input pipes* (the receiving end) and *output pipes* (the sending end)

Pipes are published and discovered using *Pipe Advertisements*, and have a unique *Pipe ID*. A Pipe offers two modes of communication: *Unicast Pipe*, which connects two pipe ends with a unidirectional and asynchronous channel, and *Propagating Pipe*, which connects one output pipe to multiple input pipes. The two communication modes are visualized in the figure below.

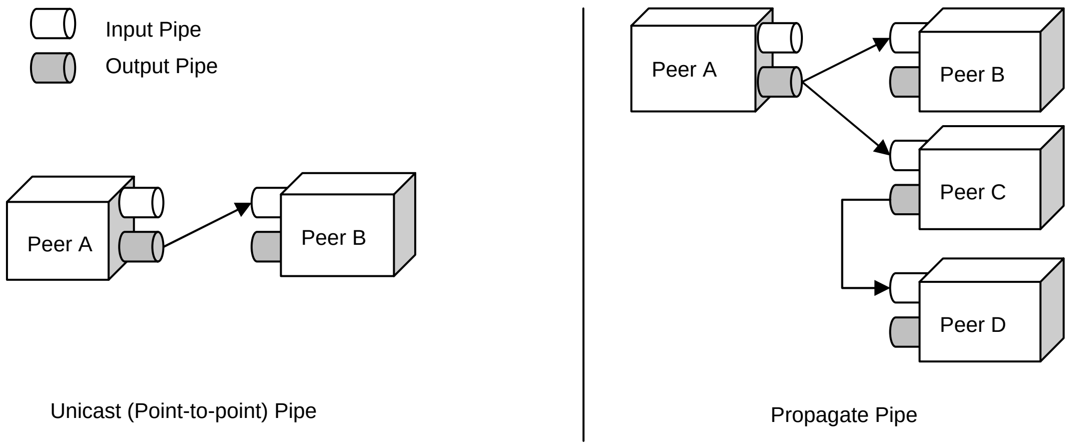


Figure 11 - Unicast and Propagate Pipes

2.5.4 JXTA Protocols

Protocols are used to dictate what data is sent and to recognize the data on a receiving peer.

Project JXTA has defined a set of six protocols divided into two categories; the *Core Specification Protocols* and the *Standard Service Protocols*. Each protocol addresses one fundamental aspect of P2P networking. The JXTA v2.0 Protocols Specification [13] describes the protocols like this:

Core Specification Protocols

- *Endpoint Routing Protocol (ERP)* is used to discover a route used to send messages between peers. If the network topology changes, the peer can use the ERP to discover new routes that are known by other peers.
- *Peer Resolver Protocol (PRP)* is the protocol by which a peer can send a generic resolver query to one or more peers, and receive one or many responses to the query.

Standard Service Protocols

- *Rendezvous Protocol (RVP)* is the protocol by which peers can subscribe or be a subscriber to a propagation service. Within a PeerGroup, a peer can be a rendezvous peer or listen to rendezvous peers. RVP is used by the PRP in order to propagate messages.
- *Peer Discovery Protocol (PDP)* is used by a peer to publish advertisements or discover advertisements such as peers, groups, pipes or content from other peers. PDP uses the PRP to send the messages.
- *Peer Information Protocol (PIP)* is used to obtain status information about peers, such as state, uptime, traffic load, capabilities, etc. PIP uses the PRP for message sending.
- *Pipe Binding Protocol (PBP)* is used by a peer to establish a virtual communication channel (pipe) between one or more peers. PBP uses the PRP to send messages.

The protocols are semi-independent of the others, so a peer can choose to implement only some of the protocols to provide functionality, and to rely on default behaviour for those not used. The next figure illustrates the use of protocols between two peers, and how the layers of the protocol stack are built to rely on each other.

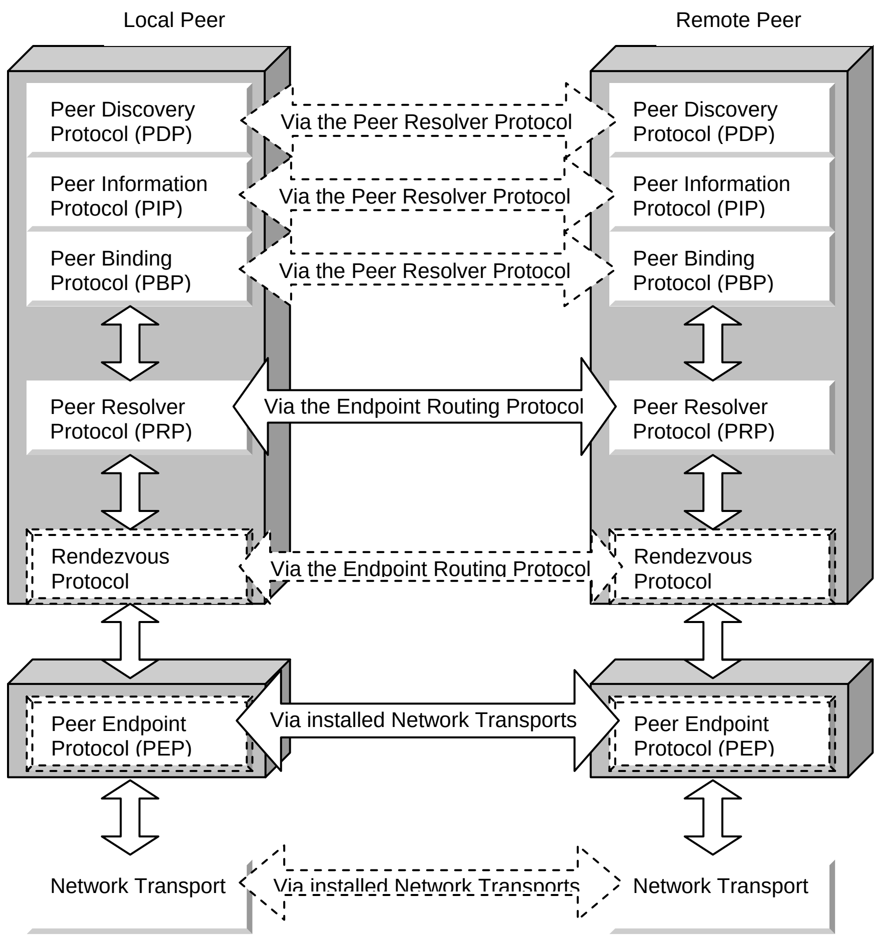

Local Peer
Remote Peer
Peer Discovery Protocol (PDP)
Via the Peer Resolver Protocol
Peer Discovery Protocol (PDP)
Peer Information Protocol (PIP)
Via the Peer Resolver Protocol
Peer Information Protocol (PIP)
Peer Binding Protocol (PBP)
Via the Peer Resolver Protocol
Peer Binding Protocol (PBP)
Peer Resolver Protocol (PRP)
Via the Endpoint Routing Protocol
Peer Resolver Protocol (PRP)
Rendezvous Protocol
Via the Endpoint Routing Protocol
Rendezvous Protocol
Peer Endpoint Protocol (PEP)
Via installed Network Transports
Peer Endpoint Protocol (PEP)
Network Transport
Via installed Network Transports
Network Transport

Figure 12 - The JXTA protocol stack.

Each protocol is divided into two parts; one part handles sending messages, the other part handles incoming messages. The protocols are written in XML, have low overhead and are easy to implement on any transport [13].

2.5.5 Advantages and Disadvantages of JXTA

The JXTA technology started out as the ultimate solution to the P2P paradigm, and the interest from the developer community was enormous. Now, after letting the hype settle, the advantages and disadvantages of JXTA become more apparent. In [12], the advantages of JXTA are defined as:

- As stated above, interoperability seeks to provide a standard way to communicate in a P2P network.
- JXTA does not limit development to a specific language, environment or networking platform.
- JXTA is available for peers behind firewalls and NATs.
- P2P functionality is provided to “every device with a digital heartbeat”, from supercomputers to digital sensors.

- XML as a way of writing messages is widely understood and compatible for the majority of platforms available. It brings the advantages of XML, such as semi structured, easy and well defined language, to the P2P developer.

However, Brendon Wilson [12] also lists some disadvantages to the JXTA platform:

- Many claim that the P2P technology needs more time to mature before developing a P2P standard. Therefore, the JXTA initiative may have come before its time.
- The extensive framework of JXTA may be too complex to learn. A developer may find it too time consuming and unnecessarily hard to keep track of its specification.
- JXTA does not attempt to address how community services are invoked. Several standards for service invocation exist, such as the Web Services Description Language (WSDL), but none has been specifically chosen by the JXTA Protocols Specification.
- The network overhead of XML messaging might be more trouble than it’s worth for small standalone applications. It might just be easier for the developer to create their own protocols if they have no intention of taking advantage of JXTA’s capability to incorporate other P2P services into the application.

All these pros and cons highlight a need for balance between flexibility and performance when implementing a P2P application. JXTA seems to be more suited for developing P2P solutions that have the flexibility to grow in the future. For a smaller, more specific P2P application, JXTA may not be efficient enough. Nevertheless, a P2P developer who uses the JXTA platform does not have to start from scratch, and can focus his work on the design of his application, not the P2P networking.

2.5.6 Summary

In this chapter I have explored the JXTA platform and given an insight to the technology, different definitions and the terminology. This terminology is widely used in this thesis, and this chapter can be helpful to understand different concepts used in my contribution. I have also listed some of the advantages and disadvantages of JXTA as a P2P Platform.

2.6 Project JXME

This chapter looks closer at the JXTA for J2ME (JXME) API, and how it can be used as a foundation for a P2P system for mobile phones.

2.6.1 Introduction

The JXTA for J2ME (JXME) Project [8] aims to provide JXTA compatible functionalities on constrained devices using the Connected Limited Device Configuration (CLDC) or the Connected Device Configuration (CDC) and the Mobile Information Device Profile (MIDP).

Using JXME, any MIDP device will be able to participate in P2P activities with other devices within the JXTA network. The project is also an open source effort by the JXTA community, and is under constant development and testing.

The first JXME implementation was done in J2ME/MIDP 1.0 to implement a compatible JXTA implementation for J2ME. Due to the constraints of the mobile devices and the J2ME platform, it is not feasible to implement a complete JXTA edge peer on a MIDP device. This is solved by moving all of the heavy workload to a *relay peer,* which is also required due to limited networking support and because the devices may be behind firewall or NAT. This relay Peer, who is called a JXME Proxy Service, runs on a JXTA Rendezvous peer. This Peer has substantial computing power, e.g. a normal desktop computer.

The figure below shows wireless devices that are members of a JXTA network connect through JXTA relays, communicating independently of underlying network protocols and network carriers. [9]

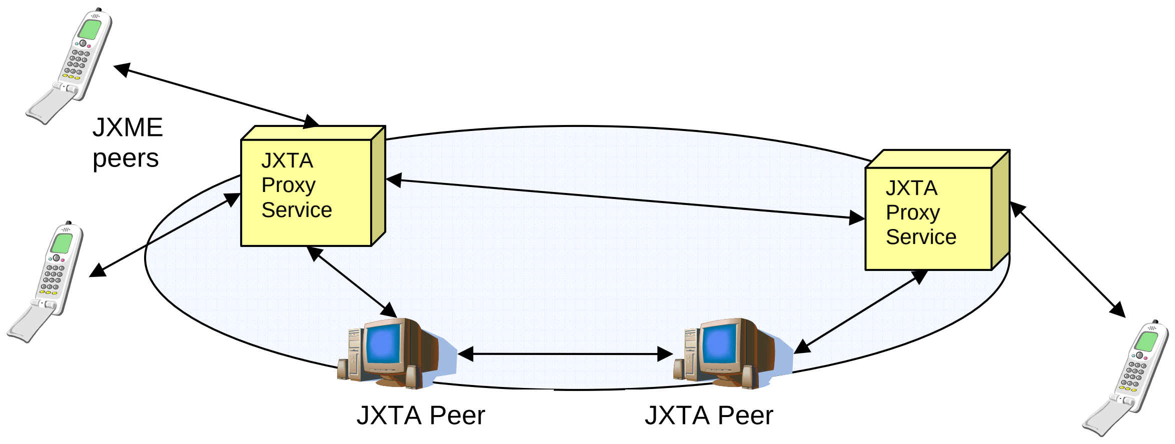


Figure 13 - JXTA Network with JXME devices

2.6.2 JXME API

The JXME API consists of 3 classes, *PeerNetwork*, *Message* and *Element*. Since the MIDP API doesn't support XML, the Message and Element classes replace this structure, giving the developer a sense of structured messaging similar to xml.

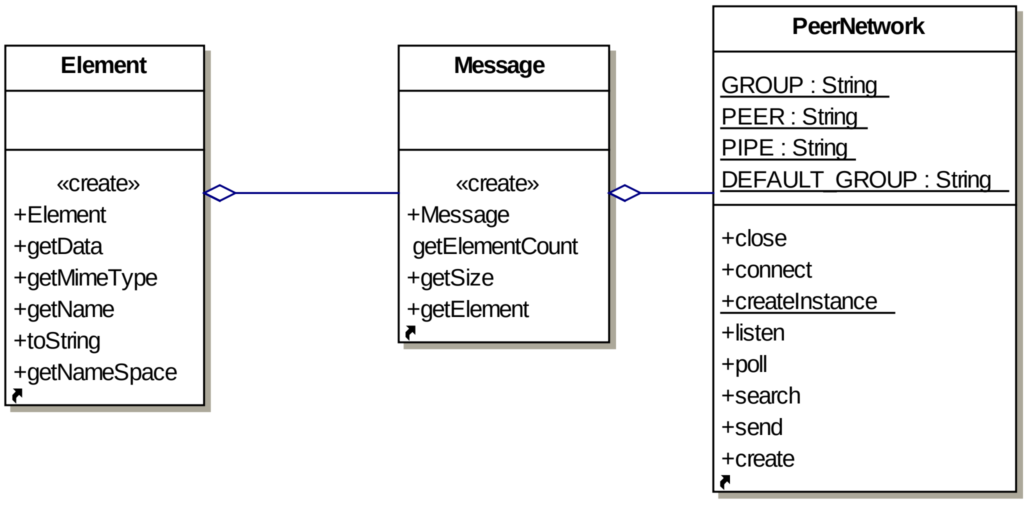


Figure 14 - JXME API

Element class

An element represents a single JXME message element, which is used by the JXME implementation to author JXME messages. This class is also used by the developer to customize his/her own JXME messages. A namespace is used to group elements, like in XML. The JXTA messages use a private namespace, namely the "jxta" namespace. The developers are free to create their own namespaces for the message elements.

The Element class takes 4 parameters:

- **name** – the name of the Element
- **data** – the data which is to be transported over the network
- **nameSpace** – The Element uses, like XML, a namespace to categorize the elements
- **mimeType** – the mime type of the data. Default is "application/octet-stream".

Message class

The message class represents a JXTA Message, which is composed of an array of elements. Certain elements are reserved for use by the JXTA network, which use the private namespace “jxta”. The Message object is sent between the relay and JXME device, and a method for traversing the Message for elements must be created on each side. The Figure to the right shows the Elements of a JXME Message.

The interesting part here is the body of the message, which describes the different elements sent between the JXME client and the JXME Proxy Relay.

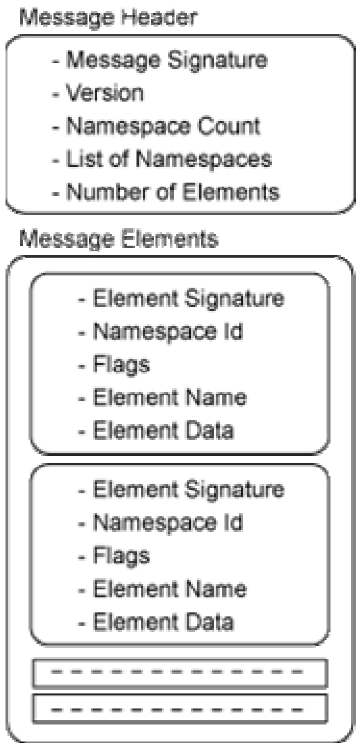


Figure 15 - Message elements

PeerNetwork class

The PeerNetwork class handles all the communication with the relay, such as connecting to the JXTA network, creating and searching for different advertisements. The PeerNetwork hides the low-level communication, giving the developer simple methods to create a JXTA Peer for the J2ME device:

- **createInstance()** .- Creates a new instance of the PeerNetwork
- **connect()** : Connects to the relay
- **create()** : Creates a new Peer, PeerGroup or Pipe
- **join()** : Asks the proxy to join a PeerGroup. Application can then use the createInstance() method to create a new instance which would be a member of the group.
- **listen()** : Opens a Pipe for input.
- **search()** : searches for Peers, PeerGroups or Pipes.
- **send()** : Method for sending data to a specified Pipe.
- **poll()** : polls the proxy for new messages.

2.6.3 JXME Proxy Service

The JXME project defines four tasks for the JXME Proxy Service [8]:

- *Compacts Advertisements*. A JXTA for J2ME peer doesn't have enough memory to store all the incoming advertisements. As a result, JXTA for J2ME relay service needs to filter unnecessary advertisements. It also strips down an incoming advertisement to the bare minimum as per JXTA for J2ME peer's requirement.
- T*ranslates Messages.* The service translates JXTA XML messages into binary messages understood by JXTA for J2ME peer and vice versa.
- *Acts as a proxy*. The proxy acts on behalf of a JXTA for J2ME peer. It
  - Creates, publishes and discovers Pipe Advertisements
  - Creates, joins and discovers Groups
- *Relays messages.* JXTA uses relays for NAT traversals. For JXTA for J2ME, a relay stores all the incoming messages for a JXTA for J2ME peer. A J2ME peer periodically polls the relay to get all the incoming messages for it. For better performance, JXTA for J2ME tries to get back data on each outbound connection, if there is any data queued at the relay for that peer.

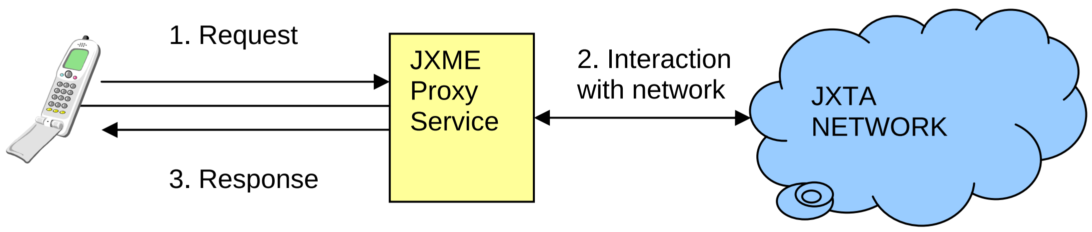


Figure 16 - The JXME proxy Service message flow

The Proxy peer belongs to the JXTA network as a normal peer, but also handles all requests to and from the mobile device in its name, just like a web proxy.

The JXME device communicates with the proxy service through a J2ME HTTP connection, and sends some predefined commands in the request headers to tell the proxy what it wants to be done. This can be found in JXTA operations like creating a pipe, joining a group, sending a message to a pipe, etc.

The proxy Service then processes the request and interacts with the JXTA network to perform the operation.

Any result from the operation is sent back to the JXME peer as a response message to the HTTP request. This can be a search result, a confirmation message or an error message. It also keeps a queue of messages for the JXME Peer that is sent when the Peer polls for messages.

2.6.4 JXME Proxyless

The next step for the JXTA development community is to create a proxy less version of the JXME implementation for CLDC devices, which removes the need for a proxy service to communicate with the JXTA network.
With the release of MIDP2.0, sockets and datagram connection (UDP transport) has been implemented for J2ME, removing the limitations of a HTTP connection and opening server functionality for mobile devices. Also, the computing power of mobile phones has evolved and will continue to improve, giving a mobile device more flexibility and peer responsibility.
At the moment, only the CDC version of JXME Proxyless is prioritized, and a beta version was released in June 2005 and is available at the project homepage [8] for download and exploration. The CLDC version depends on community effort, and the project status is not defined at the time of writing.

2.6.5 Summary

JXTA for J2ME is a project still under development and has therefore not been fully implemented and tested. The goal is to let small connected devices participate as equal Peers in the JXTA network. The project is ongoing, and is divided into a CDC implementation, for devices such as PDA's, and a CLDC implementation, for devices such as mobile phones. The JXME Proxyless API requires powerful resources, and is therefore first implemented on CDC devices. A Proxyless version for CLDC devices is still not implemented, and is depending on volunteer community effort.

2.7 Project TINI

This chapter gives a brief introduction to the TINI project, which can enable sensors to participate on a JXTA P2P network.

2.7.1 Introduction

The TINI (Tiny Internet Interface) Project [10] is also a sub-project of the JXTA community. It enables even smaller wireless devices, such as wireless sensors to participate as Peers on the JXTA network. I want to briefly describe this to show that the JXTA technology can be implemented on sensors, which is mentioned in the problem specification.

2.7.2 The TINI Binding

The TINI (Tiny Internet Interface) [10] is a Java virtual machine on a SIMM sized printed circuit board.
A sensor must have these properties:

- Ethernet interface and TCP/IP stack.
- Various I/O ports, such as 1-Wire, CAN, I2C, serial, and others
- CPU with Java virtual machine

The TINI binding for JXTA is an ongoing project to include small sensors as peers on the JXTA network. The TINI lets you Internet-enable such sensors, and JXTA gives it Peer-to-Peer capabilities. A combination gives you Peer-to-Peer appliances; small sensors contributing to a larger P2P network, or a network consisting of appliances collaborating, such as a refrigerator and a freezer to track inventory.

Another use of sensors is in medical monitoring. Body Area Networks (BAN) is a huge field of research, and enabling these as Peers in a P2P network is a huge advantage for patient monitoring over large distances.

2.8 Summary

In this chapter, I have presented some of my theoretical research in a literature study. The goal is to introduce important concepts and terminology to the reader and justify my decisions in the next chapter

3 System Design

3.1 Introduction

In this chapter I will propose a P2P system for wireless devices using the JXTA platform. The system will be based on what I have learned in my research around P2P and JXTA, and also what I have come to experience during testing – what is possible in theory and what actually works. This is closely related to the possibilities and constraints of today's mobile phones, the J2ME technology and wireless network. I will also point out what makes the JXTA framework suitable for this system.

3.2 Network Architecture

Creating a P2P System design is not easy due to its complex nature. One of the goals of the JXTA Project is to ease this heavy load off the developer, so that they can spend more time developing the actual application. Unfortunately, the JXME Project is an ongoing process and is not fully complete. Being a lightweight version of JXTA, JXME can only perform the most basic P2P operations.

I based my Mobile P2P design on my literature research and constant functionality testing. I did not want to design a system that only worked in theory; I wanted everything to be actually working, with today's technological possibilities and limitations. So, with this in mind, I have implemented many small mobile P2P applications to test different aspects of P2P, such as Peer discovery, instant messaging and file sharing. As a result, I present a Mobile P2P System based on JXME, but with my own solutions to operations not working or not supported by the JXME framework.

Even though the evolution of mobile technology is impressive, today's mobile phones still have substantial constraints which we must take into consideration when developing a mobile P2P design. As discussed in the literature study, a phone will need the help of a Proxy Service to work as an adequate peer on a JXTA network. This is also the main reason a mobile P2P network can not consist of mobile devices only. The routing and rendezvous tasks demand more computing power, memory and bandwidth than today's mobile phones can provide.

A mobile P2P system with only mobile phones as peers will need assistance from more powerful peers, like personal computers, to handle the message routing and rendezvous tasks. The mobile devices act only as leaf nodes with basic peer capabilities. The figure below shows the relationship between the physical and virtual network layer of my P2P architecture.

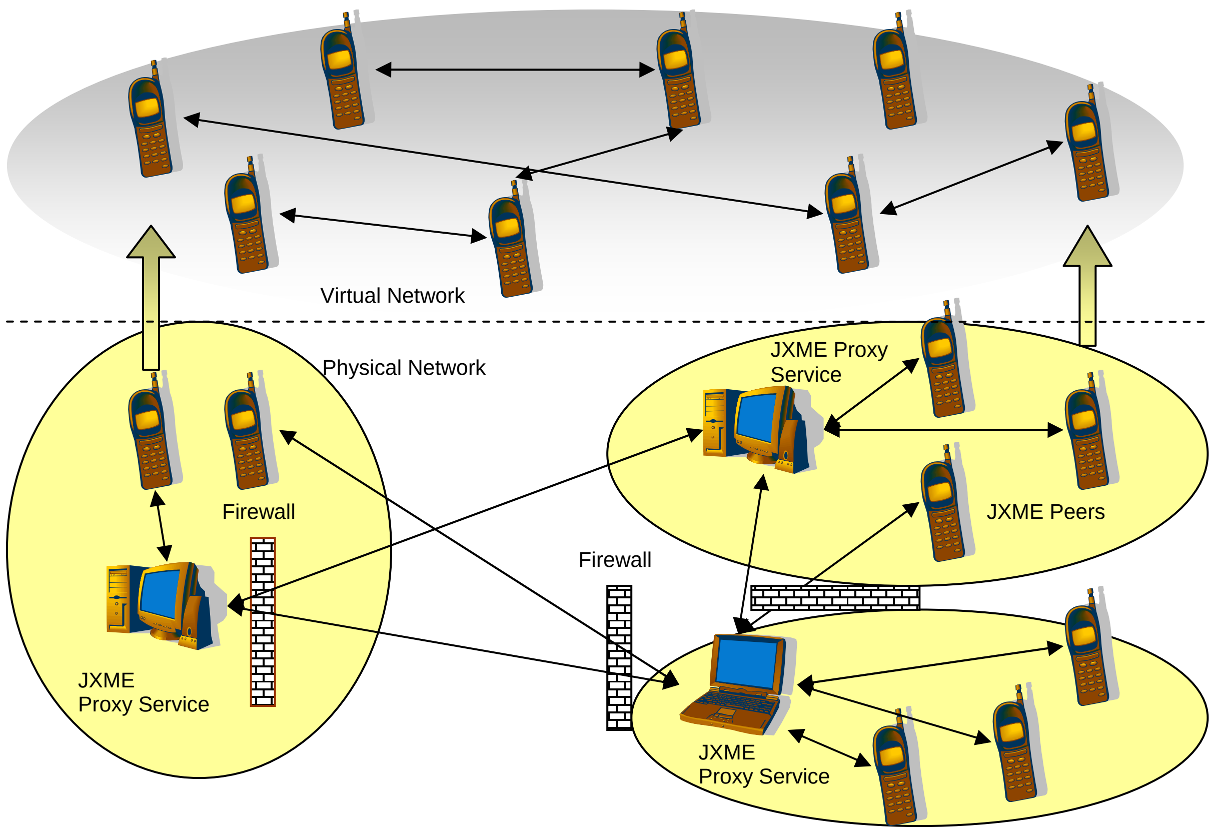


Figure 17 - Mobile P2P Architecture, physical and logical layer

Peers are nodes on the P2P network – They can have different responsibilities, and some peers are not even visible on the virtual network layer, they just relay and route messages for the leaf peers.

3.2.1 JXME Peer

A JXME Peer is a mobile phone, or a phone emulator executed on a computer. It is called a JXME Peer because it uses the JXME API to communicate with the P2P network.

The JXME Peer is always a leaf peer, with a minimum of P2P responsibilities. Due to its constraints, it must leave the heavy responsibility to the larger peers. A peer can communicate with the network in three ways:

- Send pre-defined messages to the JXME Proxy (such as connect, create pipe, join PeerGroup)
- Send a *Unicast* (direct) message to another Peer
- Send a *Propagate* (broadcast) message to all Peers on the Network

Since the JXME Peer uses HTTP Connection to communicate, it can not listen for new messages. HTTP is a request / response protocol, so the Peer needs to send a request to get data from the Proxy Service. To simulate listening, the Peer polls the Proxy Service at a given interval to check if it has any messages pending. The Poll message is an empty JXME Message.

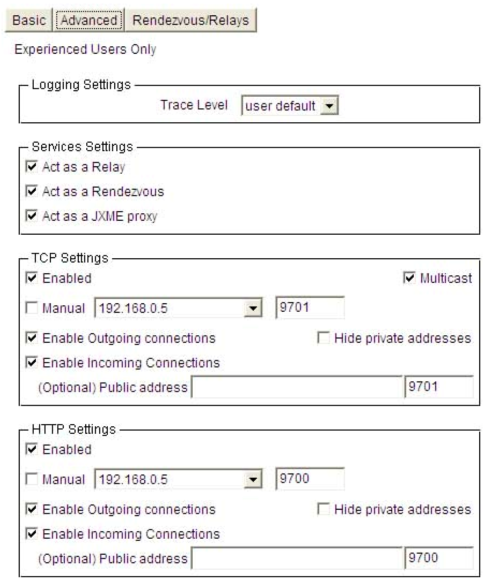
3.2.2 JXME Proxy Service

The JXME Proxy Service was described in the literature study of JXTA and acts mainly as a message relay for the JXME Peer(s). One JXME Proxy Service can be connected to many JXME Peers at the same time, and it can also have Relay-properties, which lets it communicate with the JXTA Network over a Firewall or NAT. It can connect to a Rendezvous Peer or directly to another Proxy Service.

This peer can also be a Rendezvous Peer itself, so, for a simple network, the Peer can act as a JXME Proxy Service, Relay Peer and Rendezvous Peer at the same time. This is what I propose for the JXME Proxy Service. One simple way to implement this is to use the JXTA Shell application which can be downloaded from shell.jxta.org. The working configurations for my design are explained here (JXTA-Shell version 2.3.3):

These settings enable the JXTA Shell to act as a relay, rendezvous and JXME Proxy. The IP-address is the local computer's IP, from which the JXME Peer connects to. The Port number the JXME Peer uses is in this case 9700, which is set in the HTTP settings.

On the Rendezvous/Relays tab, the "Use a Relay" checkbox should be checked.



Experienced Users Only

Logging Settings
Trace Level: user default

Services Settings
- [✓] Act as a Relay
- [✓] Act as a Rendezvous
- [✓] Act as a JXME proxy

TCP Settings
- [✓] Enabled — [✓] Multicast
- [ ] Manual 192.168.0.5 — 9701
- [✓] Enable Outgoing connections — [ ] Hide private addresses
- [✓] Enable Incoming Connections
- (Optional) Public address — 9701

HTTP Settings
- [✓] Enabled
- [ ] Manual 192.168.0.5 — 9700
- [✓] Enable Outgoing connections — [ ] Hide private addresses
- [✓] Enable Incoming Connections
- (Optional) Public address — 9700

Figure 18 - JXTA Shell Configurations

3.3 P2P Operations

In the literature Study, foundational P2P properties for a Peer were established. I will now propose a way to realize this in my design. I will divide the tasks into:

- Network Establishment
- Peer Discovery
- Advertising Resources
- Sharing large files

3.3.1 Network Establishment

The Peer connects to the JXME Proxy Service via the JXME API. It first creates a new instance of the PeerNetwork class, and then uses this to send a connect request to the proxy. After this, the peer can start performing its initial P2P operations. The messages between the Peer and the Proxy Service are illustrated in the figure below:

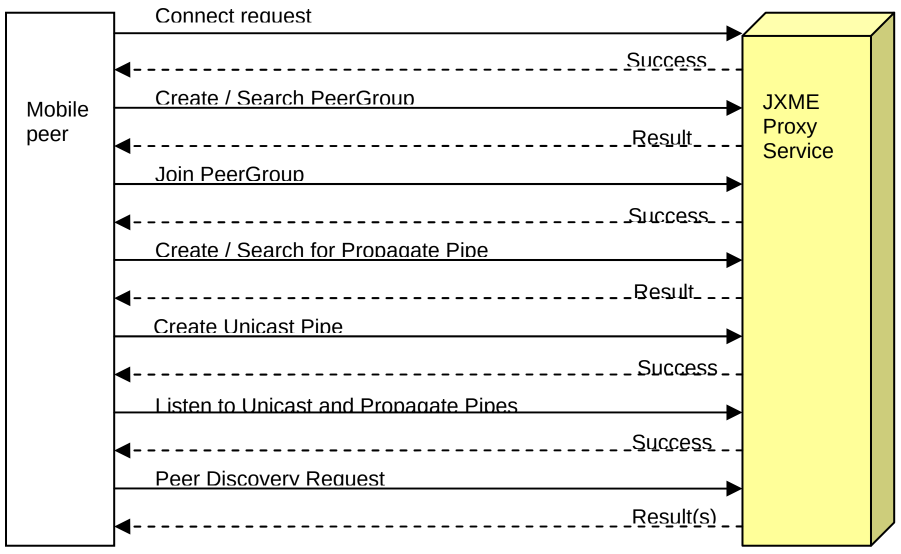


Figure 19 - Peer connecting to the network

To separate one implementation or area of interest from the rest of the JXTA network, PeerGroups are used. The founding Peer creates a new PeerGroup and joins it. Other Peers joins the existing group.

A propagate pipe is created by the founding Peer, which starts listening on it. Other Peers searches for it and starts listening. This is the common communication channel; a broadcast channel.

Unicast Pipes are created by all Peers; one unique Pipe for each Peer. This is the personal communication channel for a Peer.

To address the loss of connectivity problem, the Peer will not lose connection if it is temporarily disconnected by the phone network. As soon as it is connected again, it will be able to continue with the Peer operations on the network. All messages it may have missed during the time offline have been cached by the JXME Proxy Service, and are sent to the Peer as soon as it Polls for new messages.

3.3.2 Peer Discovery

Searching for other peers on the network with common interests can be managed in many ways. The JXME API lets you create Peer Advertisements, and search for other Peer Advertisements, by name or the id of the advertisement. After the Peer has been discovered, a search for the Peer's pipe must be executed before any communication between Peers can begin.

To minimize the network traffic, I have left out Peer Advertising from the system, only discovering Peers by the Pipes they advertise. So, when a Peer connects to a network, it creates a new Unicast Pipe, and uses this as its peer advertisement and unique ID on the network.

So, we know what to search for, but another question is how do we search for it, and how often? A mobile phone is considered to have a very unreliable connection and may connect/disconnect all the time. This is a huge issue in mobile P2P computing. Having all peers broadcasting a pipe discovery at all times would result in huge network traffic overhead, and relying on cached advertisements would not give a real-time image of the peer network. There must be a balance between the need for up-to-date data and the available network capacity.

In my research, I described three types of discovery; no discovery, direct discovery and indirect discovery. The JXTA technology aims to use indirect discovery to save network traffic, but this does not work very well in JXME, since the resource advertisements are not removed when a Peer leaves the network.

This design wants to attach importance to real-time peer discovery. Since we are dealing with mobile peers, we have a network of peers connecting and disconnecting more frequently than on a wired network. The battery may run out, the device may be out of range and so on. My proposal is to propagate a discovery request at a given interval and re-populate the peer list with the responses. The interval may change from application to application, according to their need for real-time update, and the probability of changes.

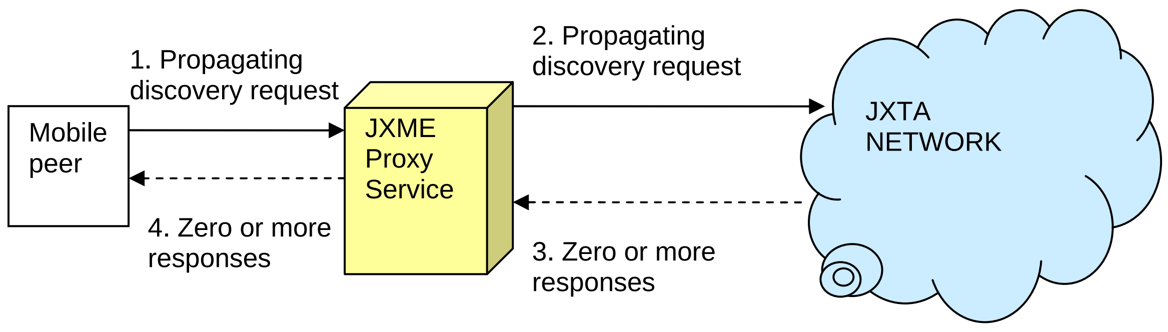

1. Propagating discovery request
Mobile peer
JXME Proxy Service
4. Zero or more responses
2. Propagating discovery request
JXTA NETWORK
3. Zero or more responses

Figure 20 - Peer Discovery

The request message is an empty message with a simple Peer-Discovery-Request header. The response message could include anything, but should be as small as possible. A pipe id and the name of the peer should be included.

The location of a JXME Proxy Service consists of an IP address and a Port number. This is the gateway for the Peer to the P2P network, so finding the JXME Proxy Service is a basic feature of a P2P application. There is no way to search for the Proxy; the Peer needs to know the address statically.

This makes the peer vulnerable to a single-point-of-failure; if the Proxy Service crashes, the peers connected to it become disconnected. A way to solve this issue is to have the address of many JXME Proxy Services, so if one crashes, the peer automatically tries to connect to another. To find the addresses of active JXME Proxy Services, a peer could check a web URL, which could be a repository for active Proxy peers, with dynamic updating.

3.3.3 Advertising Resources

The JXME API lets you create resource advertisements and search for them in the same way you create and discover Peers, PeerGroups and Pipe advertisements. The advertisement is sent to the JXME Proxy Service, and stored in its cache memory. However, this solution creates the same problem as any cached advertisements on the proxy. Once it is advertised, it stays in the cache even if the Peer has removed the advertised resource on its phone.

In my design, I let the mobile peer have some responsibility in resource advertising and discovery. The peers themselves must check their local repository and respond to a search request. This gives the peer more control, but slows down the search process, since all search messages must go out to all leaf peers.

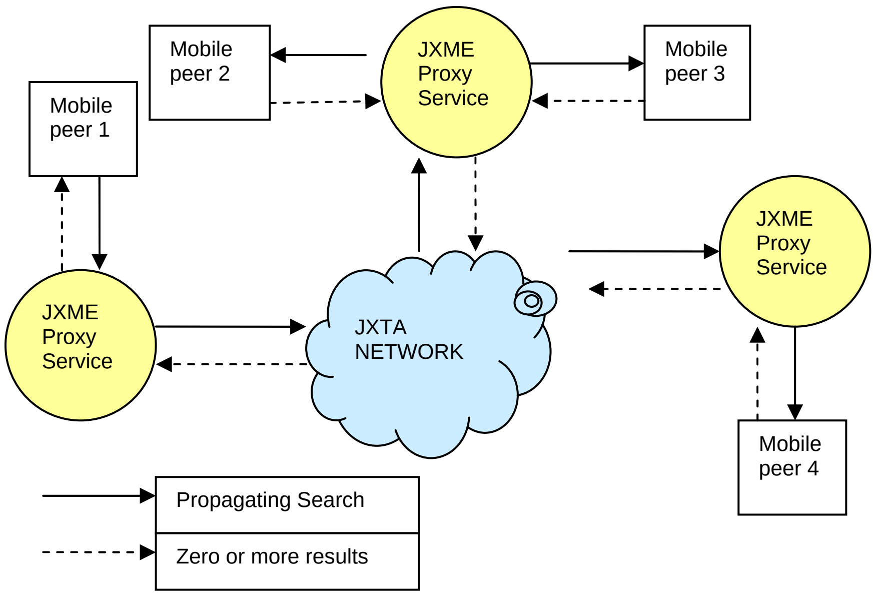


Figure 21 - File discovery

3.3.4 Sharing large files

Another aspect of mobile P2P computing is large file sharing. Sharing and downloading large files has become the most popular Peer-to-Peer activity on wired network, besides Instant Messaging of course. The main reason for this is the large bandwidth available. Mobile Peer-to-Peer on the other hand, is not suited to this. The reasons are three-folded; the *connectivity* of the mobile devices is highly unreliable compared to a wired network, the *memory* is very limited, and the *bandwidth* is low.

The bandwidth problem will depend on the network type. With the implementation of the 3G network, the mobile phone network will have capabilities to transfer large amounts of data faster than ever before. The General Packet Radio Service (GPRS), which provides moderate data speed on the GSM network, is fully functional and has a theoretical data rate of 170 kbit/s. A realistic bit rate would be closer to 30 – 70 kbit/s according to Wikipedia encyclopaedia [31].

The memory problem: Since many phones already come with memory-cards, the problem is not persistent storage, but runtime memory and available memory for a J2ME application. As discussed in the literature study, the CLDC 1.1 specification [22] assumes a minimum of 160 Kbytes of persistent memory and at least 32 Kbytes of volatile memory. Each mobile phone limits the maximum amount of data a MIDlet can store. Table 1 lists some of these properties for new Mobile Phones.

The File Connection API [35] is an optional package that many phones now contain. It allows Java to access the device file system, such as image folder or memory stick. Saving data to the file system enable the application to free the Heap memory.

The connectivity problem can be a serious issue when transferring large files between peers. This is a problem out of our control; we can only deal with it and find a solution that ensures data integrity and resends corrupt or lost data packets.

The JXME platform sends data as Messages, with elements of bytes. Unlike the JXTA API, JXME has no support for large Message segmentation. JXTA socket for example, creates a new JXTA message every time the buffer becomes full or the buffer is explicitly flushed by the application. In JXTA, the default buffer size is 16 Kb [24]. However, JXME does not implement this functionality; it sends the Message as one large stream of data. This causes problems in the JXME Proxy Service, because it can only hold a limited amount of data for each Message in its buffer. In the testing chapter, this buffer size will be investigated and tested to find the best performance.

I propose a design where large file transfer is possible for JXME devices. Once a peer has got a response to its file discovery request, it can request a file transfer from the source peer, which can start sending the file to the input pipe of the receiving peer.

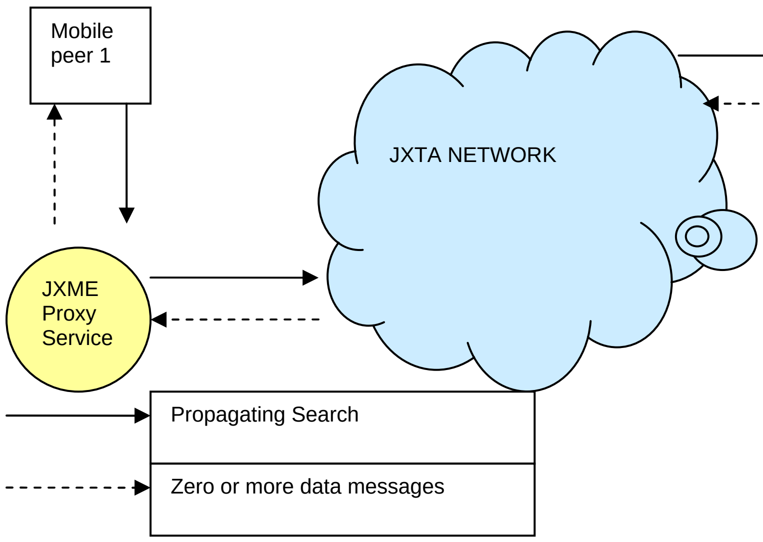


Figure 22 - File transfer request and response in the JXME Network

If a file is larger than the buffer size, the Message will be segmented and sent as chunks of smaller Messages. The receiving peer must be able to store and assemble the segmented Message in a correct manner.

If the receiver fails to receive some of the segments, the file will be corrupted. There are many ways to handle this. The easiest way – but also the slowest and most resource demanding – is to ask for the whole file again if the file is corrupted. Since a large file is segmented into many small segments, it would be desirable to just ask for the missing chunks. This can be achieved if the Messages contain info about where this chunk fits into the bigger picture – a start value and an end value.

In JXME, Elements can only contain byte streams, so all data must be converted to byte arrays. This is easy to implement with strings of text messages, but J2ME does not convert an Image class to a byte array. To transfer image files, one must preserve the image data as bytes.

3.3.5 Summary

In this chapter I have described how a P2P System can be created using JXME. The services not working in JXME or not implemented have been highlighted and an alternative way to implement this has been designed. In the next chapter, I will show such a system can be implemented in a prototype.

4 Prototype Design

4.1 Introduction

This chapter describes the design process of the prototype application. A requirement specification is extracted from a proposed disaster scenario where all user interactions are mapped and modelled using UML diagrams. Class diagrams and sequence diagrams have been modelled to show how objects interact. This work has been developed alongside the implementation process. Finally, the protocols used are listed and explained.

4.2 Scenario

"Medical Emergency and Cooperation System (MECA)"
Large natural disasters or terrorist attacks create a medical emergency which demands fast response and large supply of emergency personnel. Often the emergency is spread over a wide area and has innumerable potential victims. Such emergencies are often difficult to control and comprehend, and a need for instant communication between personnel is important; constant update of the situation and sharing of important information to map the situation and distribute help to the right areas.

With this scenario, I will propose a way to use Mobile P2P to assist in large emergencies, and I will also develop a prototype based on this scenario.

4.2.1 Preconditions

Preconditions for the scenario include a wireless connected device that the users carry around at all time, which has enabled packet based network data transport, java support and a graphical user interface. Another precondition is that the disaster area has network connectivity.

4.2.2 Goals

- Report injured people (emergencies)
- Mapping of emergency personnel
- Sharing visual information (Images) of injuries, damages to help comprehend the extent of the emergency.

- Utilize the fact that the peers are personal mobile devices, and share location information.

4.2.3 Normal action sequence

200 miles outside the coast of India the Arabian and Indian tectonic plates suddenly shift, tearing the sea floor apart. The displacement of water created by the undersea cliff causes a shockwave which is displaced into a huge wave moving through the ocean on all sides. A Tsunami.

The destruction on the coastline of Oman by the Arabian Sea is devastating, leaving a disaster area of great proportions. Thousands of houses, hotels and infrastructure are destroyed by the enormous power of the Tsunami, leaving a huge area in desperate need of help.

The international community immediately responds and several countries send rescue squads to the disaster area. A Peer-to-Peer network is established and all rescue personnel start their Medical Emergency Cooperation Application (MECA) which connects to the network.

Lisa, a volunteer from the Red Cross, arrives at the island of Masirah outside the Oman mainland and starts her MECA application. She downloaded and installed the MECA wirelessly on her mobile phone at the morning meeting and got a brief introduction to the system. After successfully logging in, the application requests a map of the area from one of the other Peers on the network. After successfully receiving the file, the main screen shows an empty satellite map. The application immediately broadcasts a message with her coordinates to all other Peers. All Peers in the rescue group for Masirah island receive the Message from Lisa, and a response message with their coordinates is sent back to her application. After a while, the map starts to populate with Peers, giving an overview of the rescue personnel distributed on the island.

After a while, Lisa enters an area clearly hit hard by the disaster. She sees many people with an immediate need for medical attention, and use the mobile to report the situation. She sends coordinates and the level of emergency, yellow, orange or red, where red is the highest level of emergency.

Soon, she senses the mobile vibrating, indicating that she has received a direct request message. The message is from another rescue person. It is a request for an image of the emergency she reported. The application has switched to camera mode, so she snaps a picture, and presses the “send image” button to send it back. She takes a new look at the map, and observers that since last time she looked, the map has been populated with several icons, indicating injured people. She can see the rescue personnel interactively moving towards the icons to help where it is most needed.

After having control of the situation in her area, she removes the emergency state from the map by broadcasting a message to remove the emergency sign. She is now available to assist in other emergencies, and checks the map. She observes many emergency signs north of her position and decides to move north to assist.

4.3 Requirement Specification

The Requirement Specification describes *what* shall be developed, not *how* it should be implemented. The requirements of this system can be divided into two parts; Functional and Non-functional Requirements. Functional requirements specify what services the system should provide, how it should react to particular inputs and events.

Non-functional requirements are constraints on the system or functions offered such as particular device constraints, look and feel, usability or security.

4.3.1 Functional requirements

This section will describe the functional requirements for the prototype based on the scenario created above. I will describe each requirement in free-text, with an identifier in the format “FR-##”, where ## is a sequential number, and illustrate the requirements with Use Case Diagrams.

I will use Unified Modelling Language (UML) to construct and visualize the artefacts of the system. Use Case diagrams are a good way to describe what the system does from the standpoint of the observer. This is closely connected to the scenario, and I will build Use Case diagrams to summarize my scenario in chapter x, and then determine requirements from this.

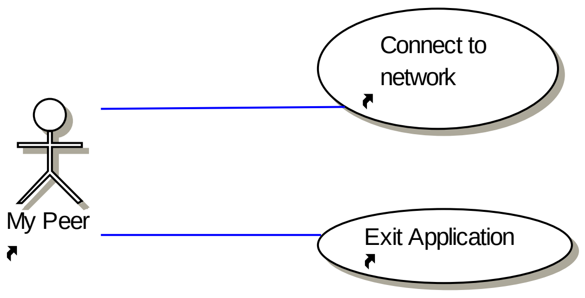


Figure 23 - Use Case 1: User Starts Application

Table 3 - Functional Requirement 01: Connect to network

FR-01: Connect to network	
Actor	My Peer
Summary	The user connects to the Peer-to-Peer network
Precondition	The application is started and the JXME Proxy Service is running
Basic course of events	1. The user sets his peer name and JXME Proxy Service URL. 2. The user executes the “connect” command 3. The application connects to the Service and joins the PeerGroup. 4. The application creates communication channels 5. A new Peer object is created (See FR-03) and the application sends the Peer advertisement to the network. 6. The Application requests the Map. 7. The application starts listening to incoming messages
Alternate paths	In step 6, if the application is a founding Peer (first Peer in the PeerGroup), then he must download the Map from a web server. If not, he requests the Map from the first Peer he discovers.
Exception paths	In step 3, the application can’t connect, and returns to the connect screen with an error message.
Post condition	The user stays connected to JXME Proxy Service
Author	Tore Mørkved
Date	19 May 2005

Table 4 - Functional Requirement 02: Exit Application

FR-02 Exit Application	
Actor	My Peer
Summary	The user terminate the application
Precondition	
Basic course of events	1. The user executes the exit command. 2. The application sends a disconnect message to the network. 3. The application is exited
Alternate paths	N/A
Exception paths	N/A
Post condition	The application is exited and memory flushed.
Author	Tore Mørkved

Date	19 May 2005

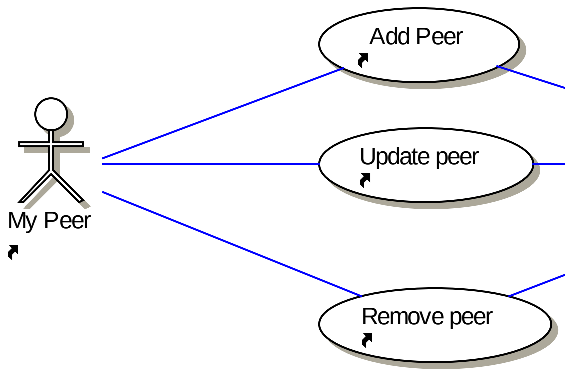


Figure 24 - Use Case 2: Peer Operations

Table 5 - Functional Requirement 03: Add Peer

FR-03 Add Peer	
Actor	My Peer, Other Peers
Summary	A new Peer object is created and added to the map
Precondition	The application is connected to network and listening to incoming messages
Basic course of events	1. My Peer connects or Peer Advertisement is received 2. Application creates a new Peer object 3. The peer is added to the map 4. The peer object is sent to the network.
Alternate paths	If the event is triggered by Other Peers, step 4 will be avoided.
Exception paths	N/A
Post condition	A new Peer is registered and painted on the map
Author	Tore Mørkved
Date	19 May 2005

Table 6 - Functional Requirement 04: Update Peer

FR-04 Update Peer	
Actor	My Peer, Other Peers
Summary	Peer information is updated.
Precondition	The peer is repainted on the map
Basic course of events	1. A peer has changed position 2. The application searches for the peer in its list 3. The application replaces the old Peer info with the new information 4. The map is repainted
Alternate paths	In step 2, if the Peer is not in the list, the peer will be added instead
Exception paths	N/A
Post condition	The Peer has moved on the map or added.

Author	Tore Mørkved
Date	19 May 2005

Table 7 - Functional Requirement 05: Remove Peer

FR-05 Remove Peer	
Actor	My Peer, Other Peers
Summary	Removes the Peer from the Peer list and map
Precondition	Peer exists in the list
Basic course of events	1. A disconnect message has been received from the network 2. The application removes the Peer from the Peer list. 3. The application repaints the map
Alternate paths	N/A
Exception paths	N/A
Post condition	The Peer is removed and no longer visible on the map
Author	Tore Mørkved
Date	19 May 2005

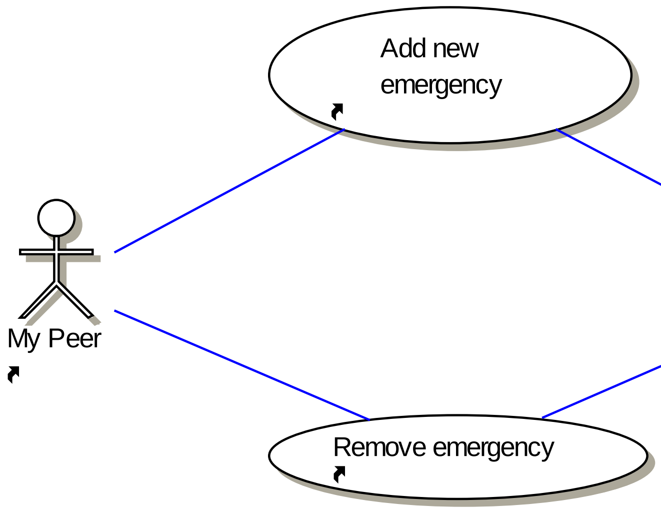


Figure 25 - Use Case 3: Emergency Report

Table 8 - Functional Requirement 06 Add new emergency

FR-06 Add new emergency	
Actor	My Peer, Other Peers
Summary	Adds a new emergency icon to the map
Precondition	My Peer is connected to the network.
Basic course of events	1. The application receives a new emergency advertisement. 2. The map list is updated. 3. The map is repainted.
Alternate paths	N/A
Exception paths	N/A
Post condition	The new emergency icon is shown in the map
Author	Tore Mørkved
Date	19 May 2005

Table 9 - Functional Requirement 07: Remove emergency

FR-07 Remove emergency	
Actor	My Peer, Other Peers
Summary	Removes an emergency icon from the map
Precondition	My Peer is connected to the network.
Basic course of events	1. The application receives an emergency advertisement with coordinates (-1,-1) 2. The emergency is removed from the list. 3. The map is repainted.
Alternate paths	N/A
Exception paths	N/A
Post condition	The emergency icon is removed from the map.
Author	Tore Mørkved
Date	19 May 2005

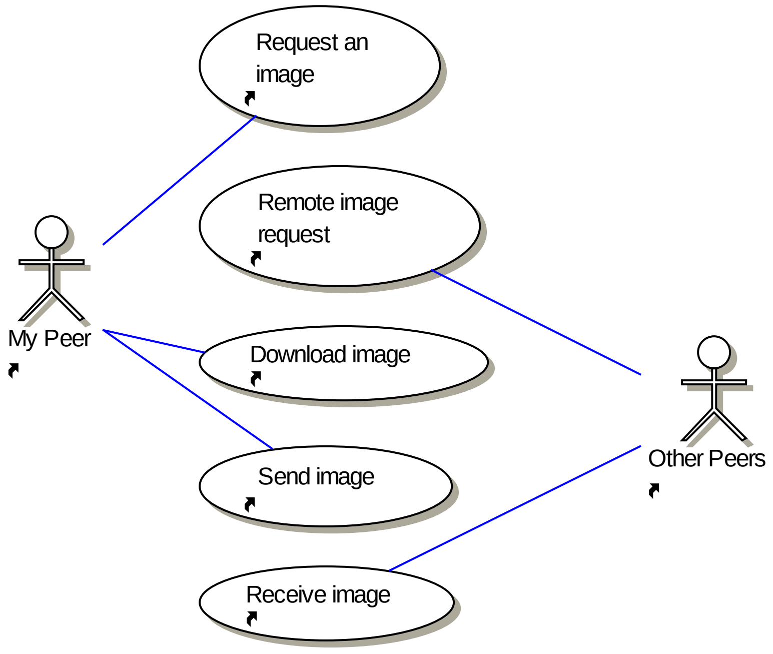


Figure 26 - Use case 4: Image Operations

Table 10 - Functional Requirement 08: Request an image

FR-08 Request an image	
Actor	My Peer
Summary	My Peer sends a request to receive an image to another Peer on the Network
Precondition	My Peer is connected to the network.
Basic course of events	1. The user writes a short message describing what kind of image he wants 2. The user selects which Peer to receive the request 3. The user executes the send command 4. The application sends the message 5. A “message sent” confirmation is shown
Alternate paths	N/A
Exception paths	If the message is not sent successfully, an error message replaces the message in step 5.

Post condition	The request message is sent to the remote Peer
Author	Tore Mørkved
Date	19 May 2005

Table 11 - Functional Requirement 09: Remote image request

FR-09 Remote image request	
Actor	Other Peers
Summary	The application receives an image request from the network
Precondition	My Peer is connected to the network.
Basic course of events	1. An image request is received 2. The application notifies the user 3. The application is set to image mode
Alternate paths	N/A
Exception paths	N/A
Post condition	The application is ready to take a picture
Author	Tore Mørkved
Date	19 May 2005

Table 12 - Functional Requirement 10: Download image

FR-10 Download image	
Actor	My Peer
Summary	The user downloads an image from a web server
Precondition	1. My Peer is connected to the network and is on the image screen.
Basic course of events	2. User enters the URL of the image file 3. User presses the “Snap!” button 4. Application downloads image from the web server
Alternate paths	N/A
Exception paths	If the image cannot be downloaded, an error screen appears.
Post condition	The image is displayed
Author	Tore Mørkved
Date	19 May 2005
Date	19 May 2005

Table 13 - Functional Requirement 11: Send image

FR-11 Send image	
Actor	My Peer
Summary	My Peer sends an image to another Peer on the Network
Precondition	My Peer is connected to the network.
Basic course of events	1. The user chooses which Peer to be the receiver 2. The user executes the send command 3. The application sends the image to the remote Peer 4. A confirmation of image sent is presented to the user
Alternate paths	N/A
Exception paths	If the image is not sent correctly, then an error message is

	shown in step 4
Post condition	The image is sent to the remote Peer
Author	Tore Mørkved
Date	19 May 2005

Table 14 - Functional Requirement 12: Receive image

FR-12 Receive image	
Actor	Other Peers
Summary	The application receives image data from the network
Precondition	My Peer is connected to the network.
Basic course of events	1. Image data is received from the network 2. The application assembles the data to an image 3. The application displays the image
Alternate paths	N/A
Exception paths	If the image is corrupt, an error message is displayed.
Post condition	The image is displayed
Author	Tore Mørkved
Date	19 May 2005
Date	19 May 2005

Summary

The table below summarizes the Functional Requirements and a priority is set to signify what is most important. “H” means high priority, and will be implemented first. “M” means medium priority, and will be prioritized next. “L” means low priority. It has the least importance and will be developed last.

Table 15 - Summary of Functional Requirements and priorities

ID	**Description**	**Priority**
FR-01	Connect to network	H
FR-02	Exit Application	M
FR-03	Add Peer	H
FR-04	Update Peer	H
FR-05	Remove Peer	L
FR-06	Add new emergency	H
FR-07	Remove emergency	L
FR-08	Request an image	M
FR-09	Remote image request	M
FR-10	Download image	M
FR-11	Send image	M
FR-12	Receive image	M

4.3.2 Non-Functional Requirements

The qualities desired for the prototype other than those concerning its functionality should also be identified. Non-functional requirements describe properties such as the applications robustness, its usability, reliability, interoperability, scalability and security. Due to the time constraints of this project, the non-functional requirements have not been prioritized. This is a time consuming part of application development, often requiring testing on real users and real usage. Nevertheless, I have tried to keep these requirements in mind when developing the application.

Usability

Usability describes the impact a system has on the end-user. In general, it refers to the efficiency with which a user can do their tasks with the product, and their overall satisfaction with that process.

My system should be easy to learn, navigate and use, so that users with no knowledge of the system should be able to use it with only a few minutes of training. The menu must be logically set up, and new messages from the network to the user must be notified using sound, vibration and/or message windows. When large network operations, such as large image transfer or downloading, are executed, interactive feedback should be displayed to the user.

Reliability

The system must be reliable, that is, it must handle and should recover from failures that may occur, such as loss of connectivity, or corrupt data transfer. This is very important when dealing with wireless connected devices. One should expect that the device will loose connectivity, even for a short time, due to power failure, if the network out is of range or other unexpected incidents.

Portability

The application should be able to run on different mobile phones, with the hardware and software requirements fulfilled. The application will be tested on emulators mainly, but if the resources are available, it will also be tested on real phones.

Security

Security is not a priority beyond JXTA security concept, considering the type of application and time constraints.

4.4 Class diagram

Class diagrams are used to describe the Classes of the system and their relationships to each other. My design recognizes three main tasks for the system; GUI interactions, network tasks and data handling. By separating these operations into different classes, we are able to change parts of the application without changing the application logic. For instance, if we want to change network connection, this can be done in the Connection class, leaving the rest of the application as it is. This type of layer programming can save time and preserves the idea of network independence.

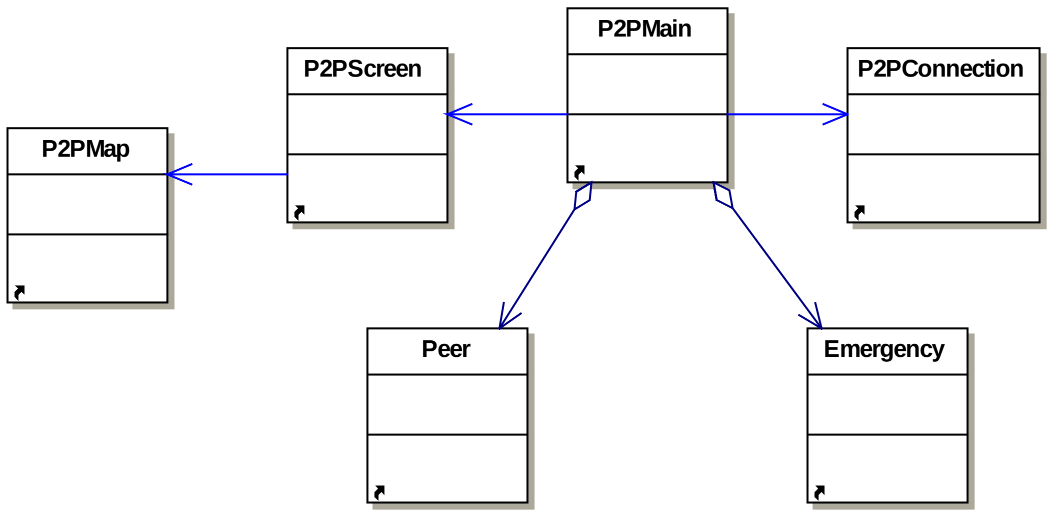


Figure 27 - Class diagram

P2PMap Class

This class extends the Canvas class, and is responsible for painting and updating the Map with emergencies and peers. It also listens for user input events, which it forwards to the P2PMain Class for handling.

P2PScreen Class

This class handles all the GUI interactions with the user, such as displaying forms, buttons and messages. It listens to user input and forwards the event handling to the P2PMain Class.

P2PConnection Class

This class handles all communication with the network, either directly or via the JXME API. All incoming messages are forwarded to the P2PMain Class for handling.

P2PMain Class

The P2PMain class is the main execution class and the engine of the application. It calls on the P2PScreen Class for all GUI operations to the user, and calls the P2PConnection Class for all communication with the network. It processes all network messages, and decides what to do in each case. The P2PMain Class also stores a list of Peer and Emergency objects.

Peer Class

Instances of this class contain all information about a Peer on the network.

Emergency Class

Instances of this class contain information about an emergency.

4.5 Sequence diagrams

To demonstrate how the classes collaborate, I have created UML Sequence Diagrams that reflect the message flow initiated by the events described in the Use Cases. For each sequence diagram, I will point out which requirements that have been fulfilled. The diagrams describe these events:

- Connect to Network
- Exit Application
- Update Peer
- Add new emergency
- Download image
- Send image
- Image request
- Receive image

4.5.1 Connect to Network

Goal: Connect to the network and perform initial Peer operations.

Fulfilled requirements: FR-01, FR-03

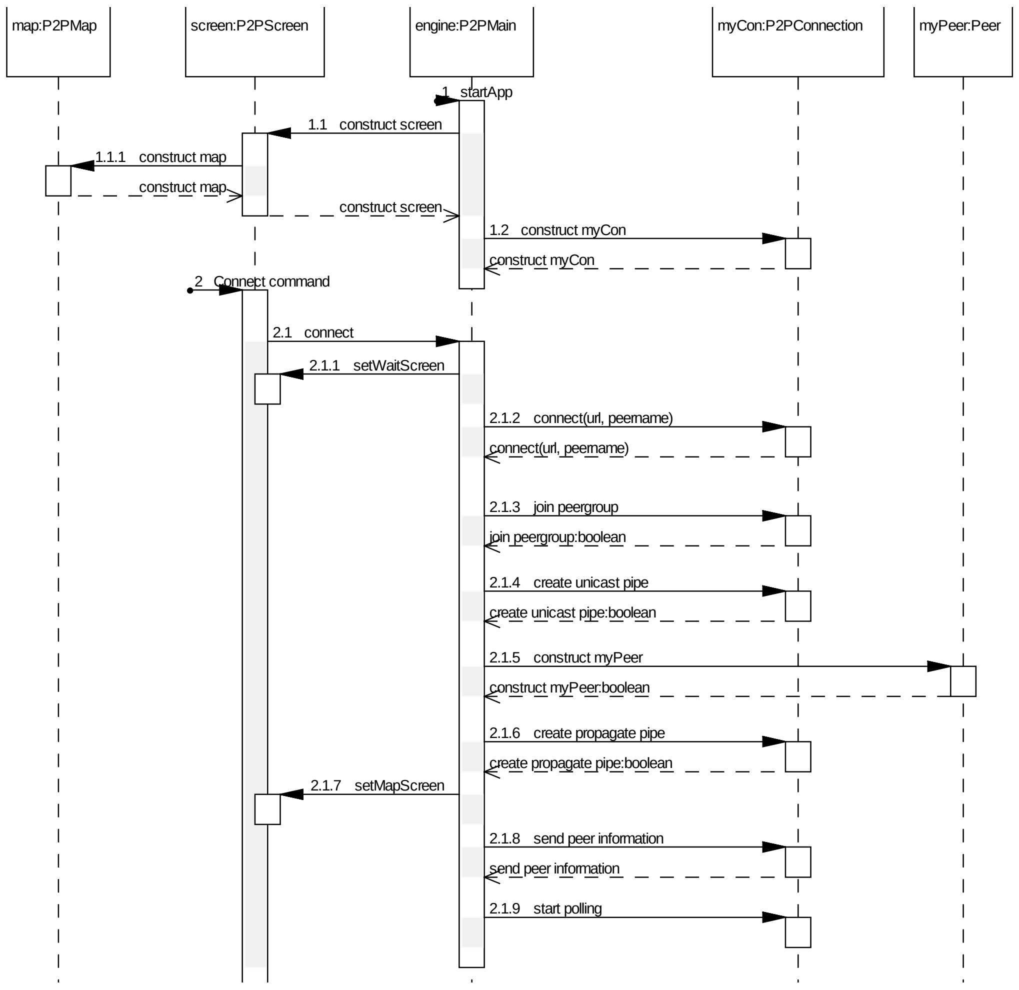


Figure 28 - Sequence Diagram, User Connect

This sequence diagram actually shows two user actions; firstly when the user starts the Application and secondly when the user chooses the connect command.

When the application executes, it constructs and initiates the different classes, and then displays the settings screen.

The connect command initiates a series of network events at the P2PMain Class to enable the Peer on the network:

1. Establish a JXME Proxy Service **Connection** to be able to send messages to the Proxy
2. **Join PeerGroup.** The peer searches for a PeerGroup, joins it and reconnects with the new PeerGroup as the active group.

3. **Creates a Unicast Pipe** and opens it for incoming messages from other Peers.
4. **Constructs** a new Peer Object to represent the user.
5. **Starts listening on Propagate Pipe** to be able to receive broadcast messages from other Peers.
6. **showMapScreen** displays the Map to the user.
7. **Send Peer Information** broadcasts own Peer information to all Peers listening on the Propagate Pipe.
8. **Start Polling.** Finally, if all steps above are successful, the application starts polling the Proxy for messages.

4.5.2 Exit Application

Goal: Add the incoming emergency to the Emergencies List.

Fulfilled requirements: FR-02

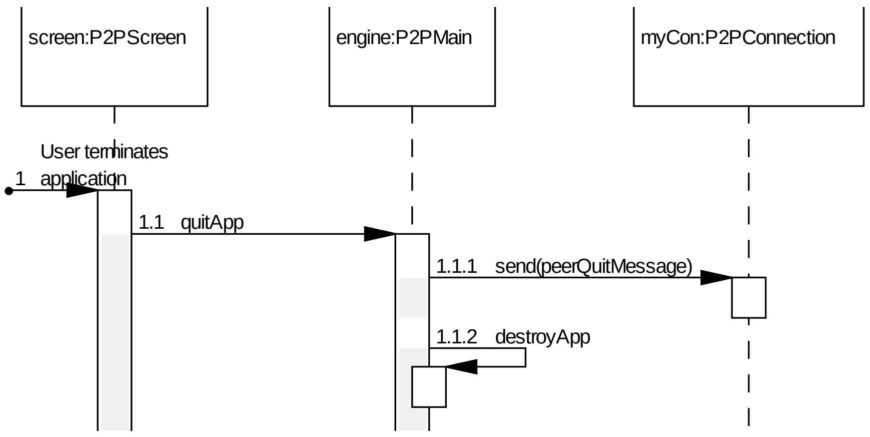


Figure 29 - Sequence Diagram, User exits

When the user terminates the application, the quitApp method in the engine is executed. Before the application closes, a Message is propagated to the network telling everyone that the peer has left.

4.5.3 Update Peer

Goal: Update peer information and display peer at new position on the Map.

Fulfilled requirements: FR.03, FR-04

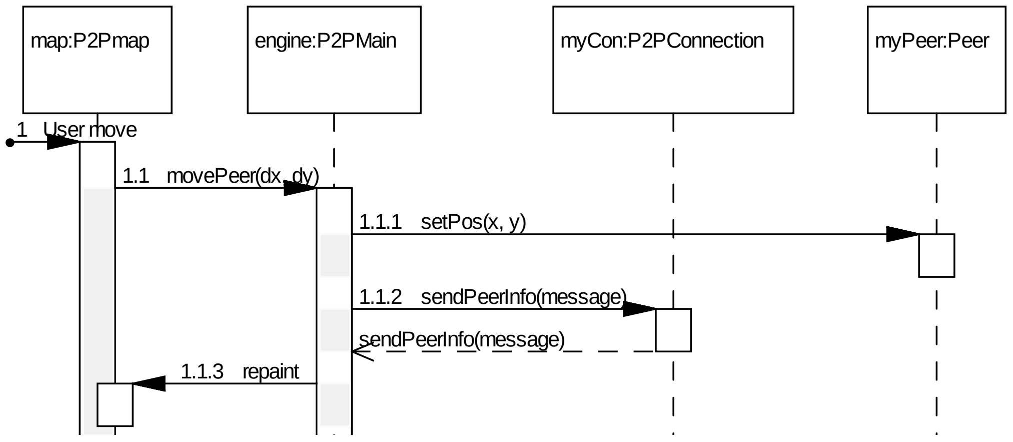


Figure 30 - Sequence Diagram - Peer Moves

When the user moves his Peer icon on the map, a movePeer message is sent to the P2PMain class, which first updates own Peer information (setPos(x,y)) and then tells the P2PConnetion Class to send a PeerInfo Message on the Propagate Pipe. Finally, the Map is repainted with the new Peer coordinates.

4.5.4 Add new Emergency

Goal: Add the incoming emergency to the Emergencies List.

Fulfilled requirements: FR-06

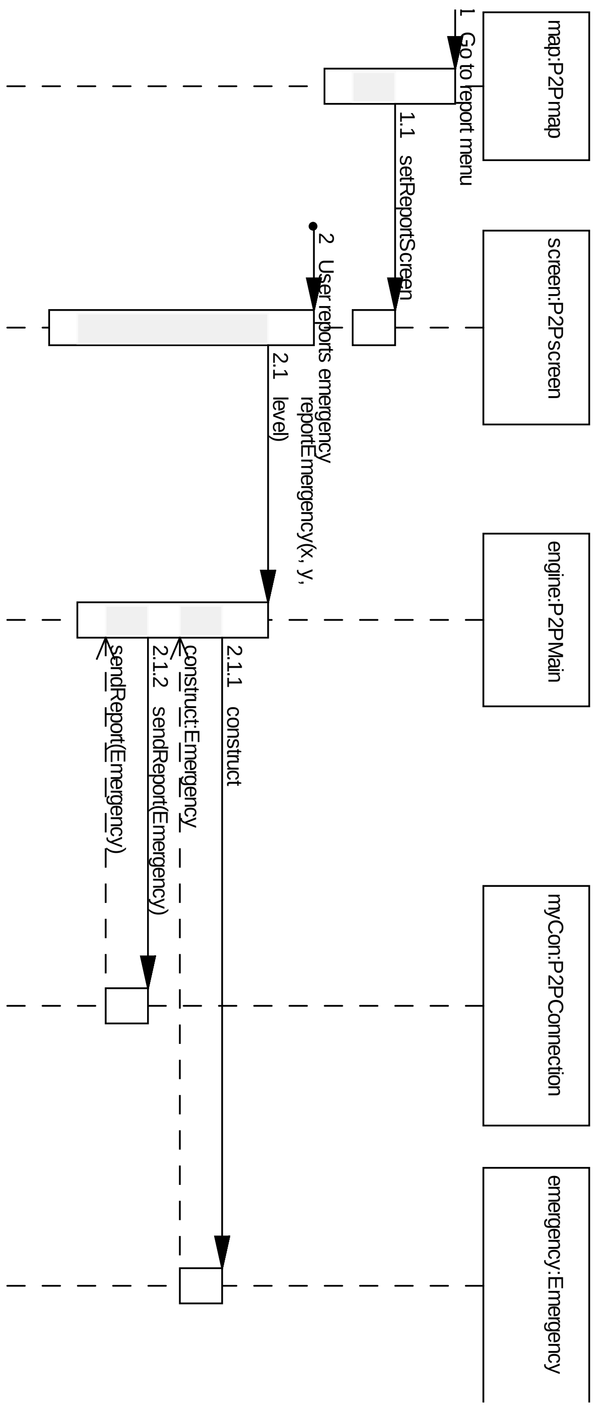


Figure 31 - Sequence Diagram - Report Emergency

When the user reports a new Emergency, the P2PScreen Class notifies the P2PMain Class, which constructs a new Emergency object, and creates a new Report to be sent. The sending is handled by the P2PConnection class.

4.5.5 Download image

Goal: Download an image from a web server

Fulfilled requirements: FR-10

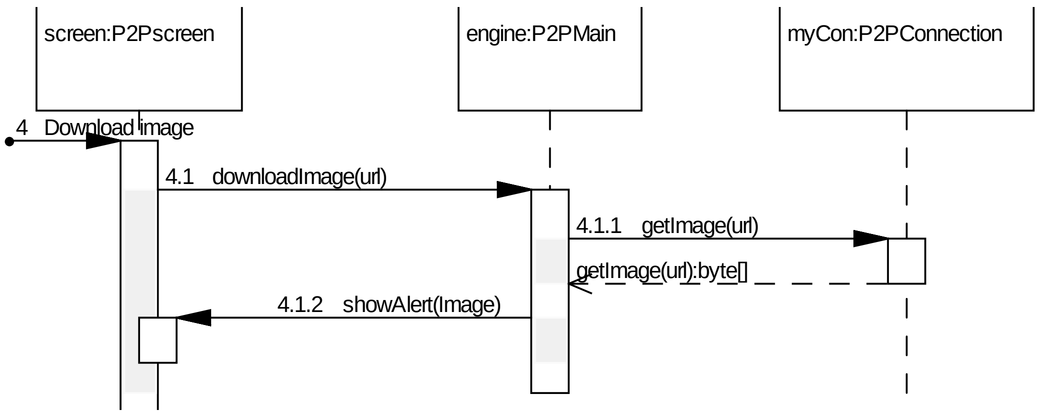


Figure 32 - Sequence Diagram, User Downloads image

This sequence is executed when the user or the system wants to download a byte stream of image data. The P2PMain asks the P2PConnection Class to download an image at a given URL and return a byte array of data. The image is shown in an Alert.

4.5.6 Send image

Goal: Sends an image to a remote Peer

Fulfilled requirements: FR-11

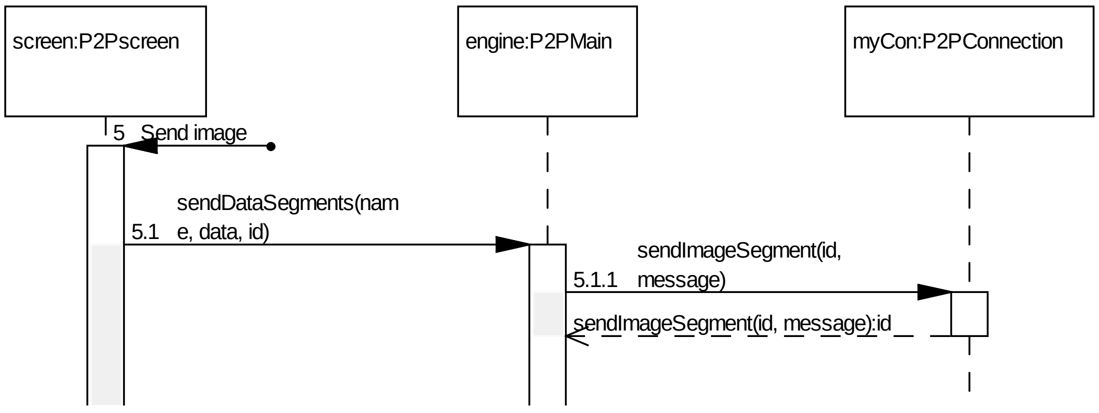

screen:P2Pscreen
engine:P2PMain
myCon:P2PConnection
5 Send image
5.1 sendDataSegments(name, data, id)
5.1.1 sendImageSegment(id, message)
sendImageSegment(id, message):id

Figure 33 - Sequence Diagram - Send Image

If the image is larger than the allowed segment size, the image is sent as a sequence of segments, which have to be re-assembled at the receiver.

4.5.7 Image request

Goal: Display image screen

Fulfilled requirements: FR-12

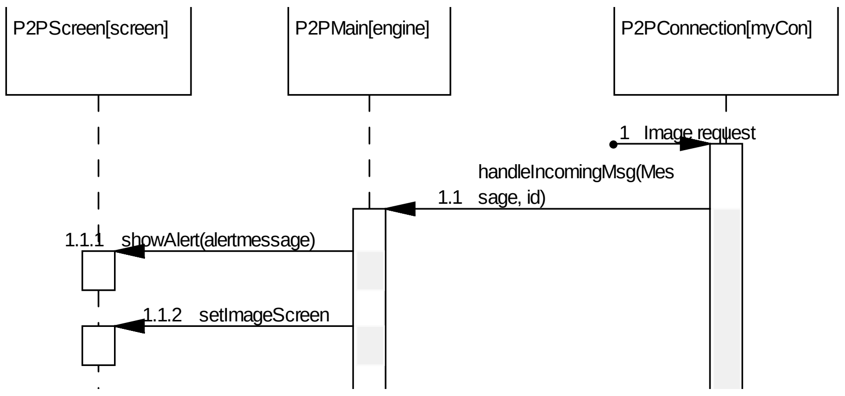


Figure 34 - Sequence Diagram, receiving image request

When the user receives an image request, the application displays an alert and then the image screen.

4.5.8 Receive image

Goal: Handle incoming image data

Fulfilled requirements: FR-12

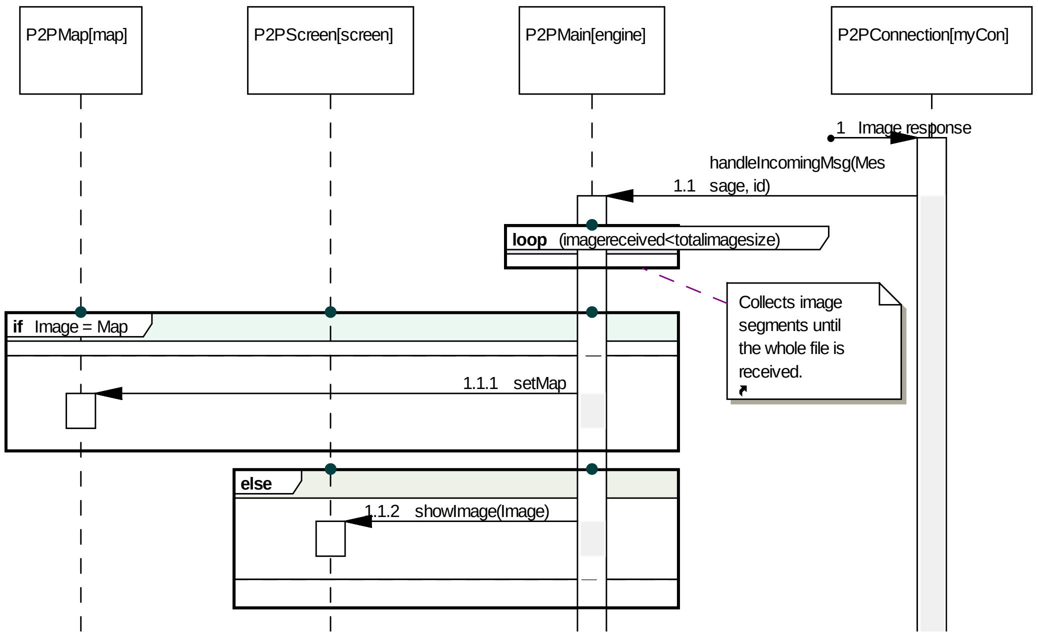


Figure 35 - Sequence Diagram, Receiving image data

When image data arrives, the application collects all data segments until the data received equals the total file size, and then displays the image. If the image is the background map, it will be set in the P2PMap class.

4.6 Protocols

My messages in this application extend the basic JXME Messages and are necessary to fulfil my requirements. These are application-specific and not a part of the JXTA core services. The messages are listed and explained below.

Table 16 - PeerDiscovery Message

Element	Value
poll	myPipeId

The PeerDiscovery Message is sent on the Propagate Pipe to all Peers listening on the network. It is a simple “poll” message, which contains the source Peer’s Unicast Pipe ID. This is used when each receiving Peer responds with the PeerInfo Message, described below.

Table 17 - PeerInfo Message

Element	Value
Peername	Name of my peer
Peerid	The pipeId of my peer
Peerxpos	The X coordinates of my peer
Peerypos	The Y coordinates of my peer

The PeerInfo Message contains all information about a Peer, and is a response message to the PeerDiscovery Message. The PeerInfo Message is sent on a Propagate Pipe after user connects to the network, and is sent on a Unicast pipe upon Peer request, as stated above.

Table 18 - ImageRequest Message

Element	Value
ImgRequest	myPipeId

The ImageRequest Message is a simple request for an image, sent on a Unicast Pipe to a specific Peer. The response to this is the ImageResponse Message, stated below.

Table 19 - ImageResponse Message

Element	Value
ImgResponseName	Name of image to be transferred
ImgReponseData	The data of the image segment
ImgResponseStart	Startvalue of image segment
ImgReponseEnd	End value of image segment
ImgResponseSize	Size of the entire image file

This Message contains image name, data and segment information. If the image is larger than allowed message size, the image is segmented and sent as data chunks. By using the last three elements, the receiver can re-assemble the image segments into one large image again in a correct manner.

Table 20 - Report Emergency Message

Element	Value
reportX	X coordinate of reported emergency
reportY	Y coordinate of reported emergency
reportLevel	Level of emergency

The Report Message is simply a message sent to all Peers with information about a new Emergency. The X and Y are map coordinates, and reportLevel is the level of emergency, ranging from index 0 to 2, where 0 is the highest. A value of -1 indicates that the emergency is to be removed.

Table 21 - Remove Emergency Message

Element	Value
reportX	X coordinate of removed emergency
reportY	Y coordinate of removed emergency
reportLevel	-1

When users want to remove an emergency, they send the same Message as when they report an emergency, but the level element value is -1, indicating that the emergency should be removed.

Table 22 - Map Request Message

Element	Value
mapRequest	My Pipe ID

A request sent to a Peer asking it to transfer the background map. The response is sent as an Image Response Message with image name “Map”.

4.7 Summary

This chapter has presented the prototype design, which is based on the System Design in chapter 3. All requirements for the application have been extracted from a proposed scenario. Next chapter will present the implemented application and test documentation.

5 Performance Results

5.1 Introduction

The Prototype was implemented in Java using the J2ME Wireless Toolkit 2.2 [31], which is very helpful when developing mobile applications. It emulates a mobile device and offers tools to compile and test MIDP applications.

In this chapter the developed application is described through a user interface walkthrough and tested regarding to functional requirements, network and large file transfer.

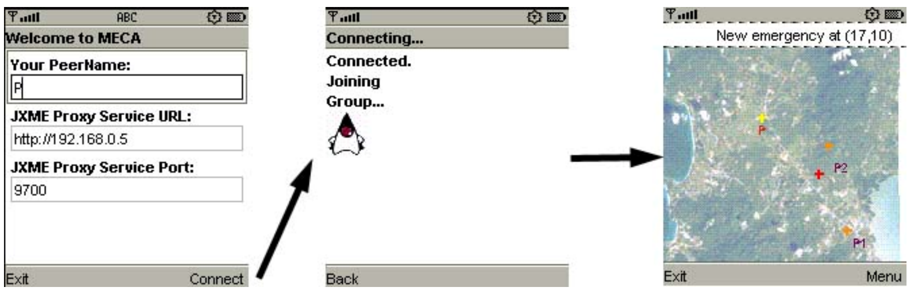
5.2 User Interface

The prototype is a simple MIDlet with few screens. The options for the user are mainly limited to reporting or removing emergencies and requesting image transfers. To better understand the application, a visual representation of the User Interface will be presented in a simple walk-through of the application.

The first screen the user meets after starting the application is the settings screen. The user enters his Peer name and the IP-address and Port number of the JXME Proxy Service it wants to connect to, and then presses the “Connect” button.

This executes a series of network events deeply described in the prototype design. Sending messages over the network is more time demanding than other events, and it is therefore important to give the user some feedback while the application is communicating with the P2P Network. A wait screen displays an icon and some information text to let the users know what is going on, giving them a sense of progression.

After connecting, the map image is requested. If this is the Peer starting the group, the image will be downloaded from a web server over a HTTP connection. If the Peer is not an initiator of the network, a request to transfer the map is sent to the first Peer discovered.


Welcome to MECA

Your PeerName: P

JXME Proxy Service URL: http://192.168.0.5

JXME Proxy Service Port: 9700

Connecting...

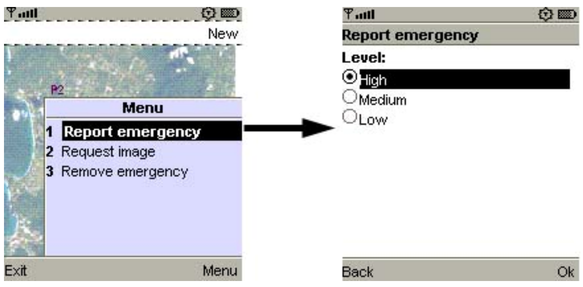
Connected.
Joining
Group...

New emergency at (17,10)

Figure 36- Screenshots: User connects

After successfully connecting and downloading the map, the application displays the map screen. This screen is continuously updated when new Peers or Emergencies are discovered, updated or removed. At the top of the screen, a scrolling ticker displays miscellaneous information. On the menu bar, there are three alternatives, namely “Report Emergency”, “Request image” and “Remove emergency”. The “Exit” button sends a disconnect message to the network and then terminates the application.

Report emergency

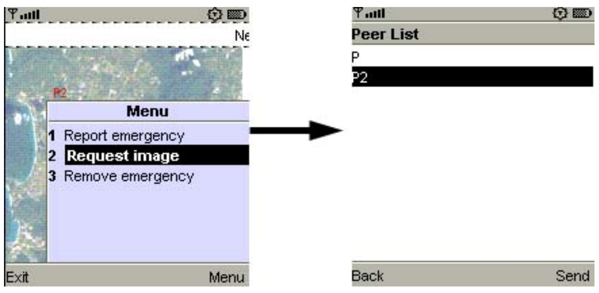

Report emergency

Level: [✓ High] Medium Low

Figure 37 - Screenshots: Report emergency

Pressing the Report Emergency button displays the Emergency screen, where the user can choose the level of emergency, and then send the report on the Propagate Pipe. After sending the report, the Map screen is displayed again, now updated with the new emergency.

Request image


Peer List

P
[✓ P2]

Figure 38 - Screenshots: Request image

Pressing the Request image button displays a list of known Peers. The user chooses the Peer he wants an image from, and sends the image request message by pressing “Send”.

Remove emergency

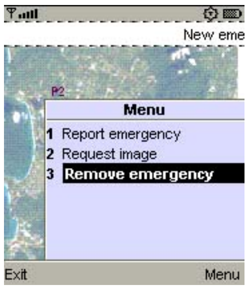
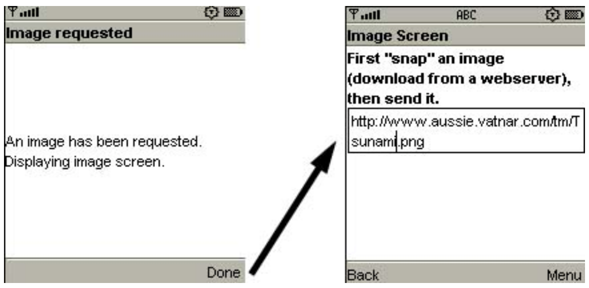


Figure 39 - Screenshots: Remove emergency

Pressing this button sends a message to the network to remove an emergency. The coordinates are taken from the Peer’s position.

Receiving image request



An image has been requested.
Displaying image screen.

First "snap" an image (download from a webserver), then send it.

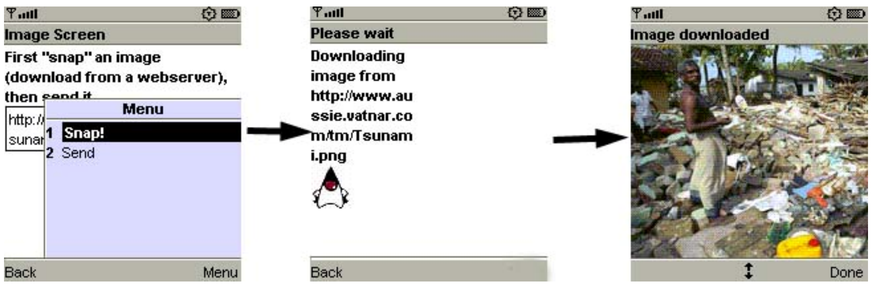
http://www.aussie.vatnar.com/tm/Tsunami.png

Figure 40 - Screenshots: Receiving image request

If users receive an image request, the application displays an alert, telling them someone requests him to snap an image and return it.

After the user presses the “Done” button, the Image screen is displayed. There are two menu choices here; “Snap!” and “Send”.

Snap an image (Download)



First "snap" an image (download from a webserver), then send it.

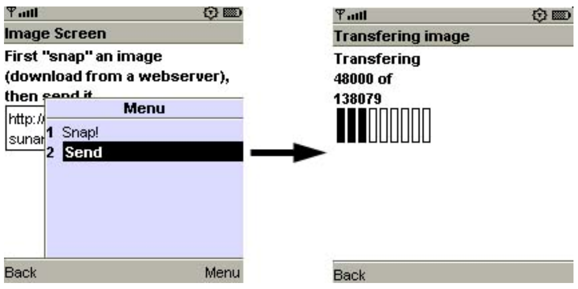
Downloading image from http://www.aussie.vatnar.com/tm/Tsunami.png

Figure 41 - Screenshots: Downloading image

“Snap!” substitutes the camera-function, simulating snapping an image by downloading an image from a web server. The intended idea would be to invoke the camera on the mobile device, but this was not implemented due to time constraints.

When the image is downloaded, it is displayed to the user before the Image screen is displayed again.

Send



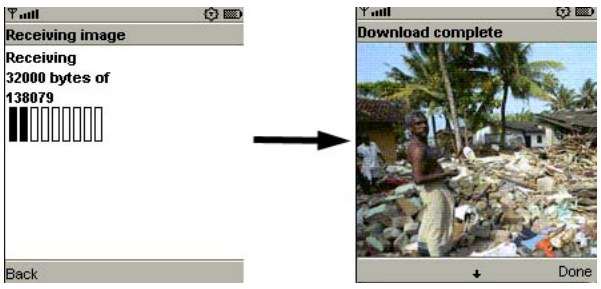
First "snap" an image (download from a webserver), then send it

Transfering
48000 of
138079

Figure 42 - Screenshots: Sending image

The second choice is "Send", which sends the downloaded image to the requesting Peer. If the image is larger than the buffer size, the image will be segmented and sent as smaller data-pieces. On the receiving side, the data-pieces are re-assembled and put together to an image. To give the user, both the sending Peer and the receiving Peer, a sense of progress, the user interface displays a progress bar informing the user how many bytes are sent or received.

Receiving image



Receiving
32000 bytes of
138079

Figure 43 - Screenshots: Receiving image

When a Peer receives an image, a progress bar shows the progress of the download. This is for demonstration purposes only; a better solution would be to let the download continue seamlessly in the background. After all the image pieces are received by a Peer, the image is shown in an Alert display.

5.3 Application Source Code

The source code of the prototype is found in the CD attached with this report. The code contains comments describing the program logic.

5.4 Functionality Test

This test focuses on the program logic according to the functional requirements.

5.4.1 Testing environment

Functionality tests have been performed between two Peers and a JXME Proxy Service on a local computer. This is a good environment to test program functionality, but not to test the P2P network.

5.4.2 Test results

I have set up a simple test document for the functionality testing where test cases and desired results have been added along with the implementation. The test cases follow the Use Cases and functional requirements set for the prototype to detect possible errors or deviations from the desired behaviour of the system.

Table 23 – Test, Use Case 1: Connect to network

No.	Test case	Desired result	Result
T-1.1	Connect as a founding Peer (P).	The “Map” screen appears, showing a map image and the Peer icon at a random spot on the Map.	OK
T-1.2	Connect as a joining Peer (Px).	The “Map” screen appears, showing a map image and the Peer icon at a random spot on the Map.	OK
T-1.3	User presses the Exit button	The application closes.	OK

Table 24 – Test, Use Case 2: Peer Operations

No.	Test case	Desired result	Result
T-2.1	Add new Peer object	Add new Peer to the Peer List, which is displayed on the Map at given coordinates.	OK
T-2.2	Update Peer object	Update the Peer with new coordinates and show the changes on the Map.	OK
T-2.3	Remove Peer object	Delete Peer from Peer List and not show the Peer anymore on the Map	OK

Table 25 – Test, Use Case 3: Emergency Report

No.	Test case	Desired result	Result
T-3.1	Add new emergency	New emergency added to the Emergency List and displayed on the Map with given coordinates and emergency level	OK
T-3.2	Remove emergency	Remove emergency at given coordinates.	OK

Table 26 – Test, Use Case 4: Image Operations

No.	Test case	Desired result	Result

T-4.1	User presses the “Request image” button	The application displays a Peer List and a “send” button	OK
T-5.2	User chooses a Peer and presses the “Send” button	An image request message is sent to the selected Peer.	OK
T-5.3	User receives an image request from a remote Peer	Application displays an alert on the screen and the phone vibrates. After the alert, the Image screen is displayed	OK
T-5.4	User chooses an image URL and presses “snap!” button.	The Wait screen is displayed until the image is downloaded. Then, an alert showing the image is displayed.	OK
T-5.5	User has done T-2.4 and presses “Send” button	Application starts sending the downloaded image and displays a progress bar showing how many bytes have been transferred. When finished, Map screen is displayed.	OK
T-5.6	User has not done T-2.4 and presses “Send” button	An information alert tells the user he must snap a picture first.	OK
T-5.7	An error occurs when trying to download image	An error alert is displayed to the user, and then return to the Image screen	OK
T-5.8	User receives an image	Application displays a progress bar showing bytes downloaded. After successfully downloading image, it is displayed.	OK

5.4.3 Summary

The application works within the range of functional requirements, but it has not been tested properly for exception handling and has not taken every possible combination of events into consideration. This is just a prototype application, and some limitations had to be set because of the time constraints of the project.

5.5 Network test

5.5.1 Testing environment

During most of the implementation process, the network is often created and tested on a local host, with only one JXME Proxy Service and a number of Peers. To test the scalability, a larger network has been set up with different Proxy Services on different hosts, with many Peers connected to each of these Proxy Services. First of all, it would be interesting to see that this works in practice, but secondly we should examine what happens when we drastically increase the number of Peers. It is difficult to measure network performance in such a network. My primary goal is to set up large networks with many Peers, and see how the P2P application performs regarding to Peer discovery, if and how fast the Map is updated when the Peers interact by moving around or reporting emergencies.

5.5.2 Test results

Table 27 - Network Test 1

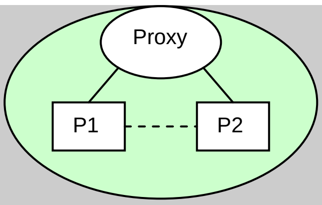
Test ID	**NT-01**
Condition	One JXME Proxy Service and two Peers on same host.
Illustration	 **Figure 44 - Network test 1**
Performance results	All Peers found each other and communicated. Very low network delay.

Table 28 - Network Test 2

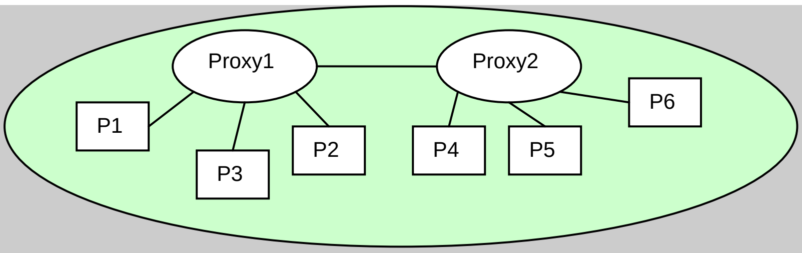
Test ID	**NT-02**
Condition	Two JXME Proxy Services on same host, three Peers connected to each Proxy Service.
Illustration	 **Figure 45 - Network test 2**
Performance results	All Peers connected and discovered by other Peers. All Peer operations were updated on other Peers within 1-2 seconds.

Table 29 - Network Test 3

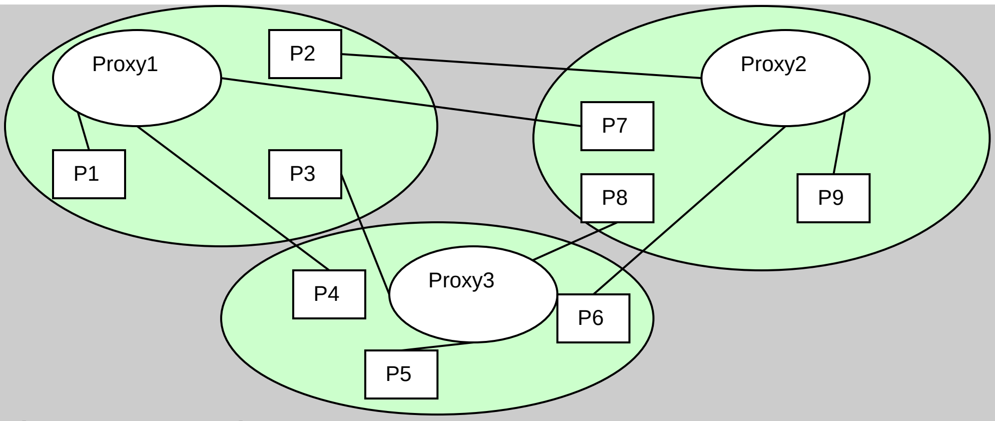
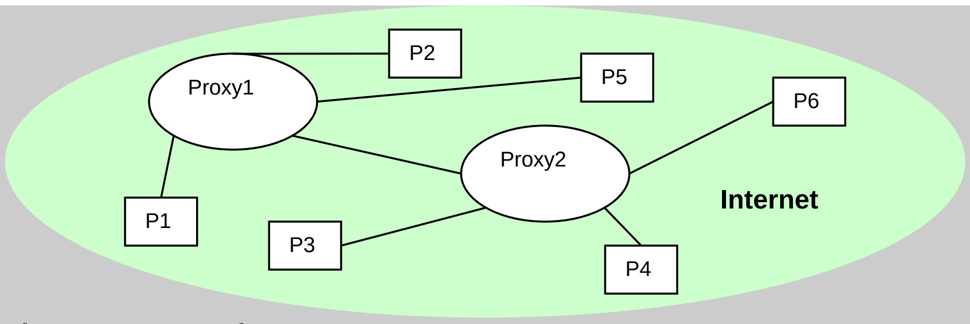
Test ID	**NT-03**
Condition	Three different hosts on a local area network (LAN) with one JXME Proxy Service on each host. Three Peers on each Proxy Service, all from different host machines.
Illustration	 Proxy1 P2 Proxy2 P7 P1 P3 P8 P9 P4 Proxy3 P6 P5 **Figure 46 - Network test 3**
Performance results	All Peers were connected and discovered. However, not all messages are received on all Peers at all times. Randomly, some of the Peers do not always update Peer movement or new Emergencies. The rate of this is not extensible tested, but from this test about 1 of 10 operations failed on about 1 of 10 Peers.

Table 30 - Network Test 4

Test ID	**NT-04**
Condition	Three hosts setting up a JXME Proxy Service at different locations on the Internet. Increasing number of Peers
Illustration	 P2 Proxy1 P5 P6 Proxy2 Internet P1 P3 P4 **Figure 47 - Network test 4**
Performance results	Peers are able to connect to a remote Proxy on the Internet. The Proxies are able to discover each other and share advertisements. All Peers are able to discover each other and share information.

5.6 Testing large file transfer

To check the transfer speed of image data and optimize the data transfer rate it would be interesting to measure transfer speed at different criteria. The influencing variables are the *size of the file*, the amount of data sent in each segment (*buffer size*) and also the *network topology* (routing, bandwidth).

I created a test-version of my Prototype for this purpose, which transferred an image between two Peers many times, and each time with an increased buffer size. The program timed each transfer – from transfer start to the peer got a response that the whole file was successfully received – and then printed the result in seconds.

5.6.1 Testing environment

The testing will be performed in a sequence of image request / response messages between two Peers running the application. The major testing will be performed with both Peers and JXME Proxy Service running on a local computer. This will reveal the JXME Proxy Service's ability to forward large amounts of data. We will also test if there is a large difference if we have a more distributed network with Peers at different hosts on the internet.

5.6.2 Variables

Buffer size is the most interesting variable to change, since this decides how much data is sent in each segment. The JXTA specification implies that the upper limit of data a relay Peer, such as the JXME Proxy Service, can handle in each Message is 60 kB. Even though the limit of a Message is this high, we want to keep the Message size as low as possible to avoid long processing time with the Relay, which should be able to handle multiple requests simultaneously.

So, what we are looking for is a small data segment size that results in fast file transfer. The size of the file should set between 50 kB and 150 kB to give a reasonable testing scenario without exceeding the memory limits of a MIDlet. I will perform the tests on two files, the first one is 73 kB and the second is 138 kB. I will increase the buffer size for each test, and perform each test three times to ensure the test results are reliable and not affected by single-events such as random computer memory blocks or network congestion. All in all, both files were transferred 21 times during the tests.

5.6.3 Test results

The test results are presented in a data table, with Buffer size in kilo bytes and transfer time in seconds. To better interpret the results, I used MS Excel to create a diagram, drawing a graph for each test.

The results will be analyzed and discussed in the Discussions chapter.

Test ID: FT-1

Buffer size: 4 – 52 kilobytes, with interval of 4 kilobytes

File size: 73 kB

Table 31 - File-transfer Test FT-1

Buffer (kB)	Test1 (sec)	Test2 (sec)	Test3 (sec)
4	117	112	115
8	65	71	73
12	50	44	53
16	40	44	44
20	28	36	33
24	33	36	37
28	31	27	29
32	31	31	29
36	36	31	32
40	28	25	27
44	26	26	30
48	28	30	30
52	30	31	32

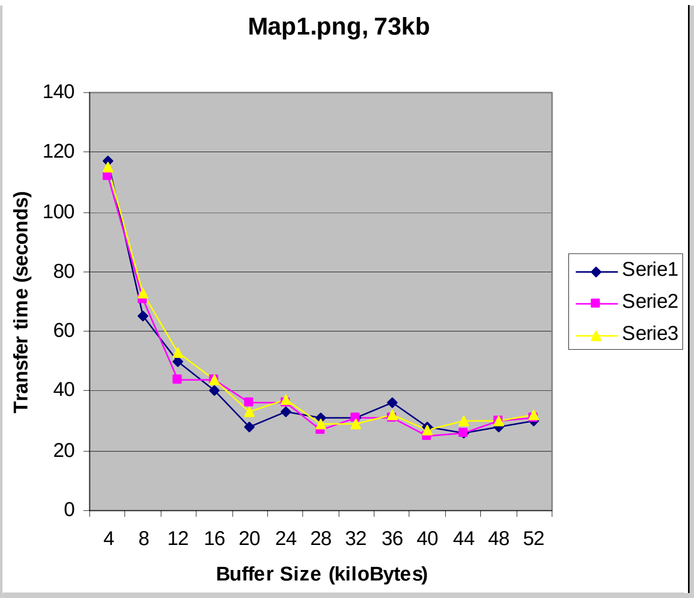


Figure 48 - Diagram of FT-01

Test ID: FT-2

Buffer size: 4 – 52 kilobytes, with interval of 4 kilobytes

File size: 138 kB

Table 32 - File-transfer Test FT-2

Buffer (kB)	Test1 (sec)	Test2 (sec)	Test3 (sec)
4	199	201	200
8	129	102	119
12	84	78	81
16	65	68	71
20	61	62	59
24	48	53	55
28	47	42	47
32	49	47	47
36	42	42	41
40	50	47	49
44	51	51	50
48	42	43	44
52	46	42	45

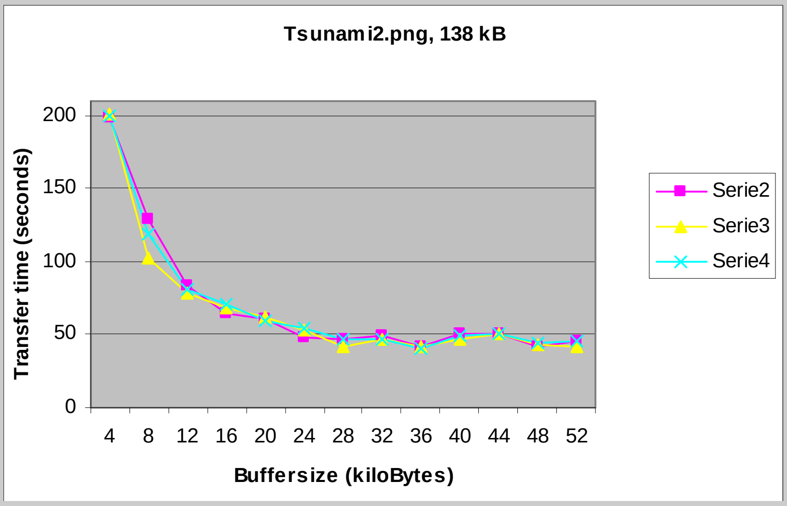


Figure 49 – Diagram of FT-02

5.7 Summary

This chapter has presented three test documents for the prototype application; functionality test, network test and large file transfer test. The results will be analyzed and discussed in next chapter; the discussion.

6 Discussions

6.1 Introduction

In Chapter 2, different technologies were presented. They were used to design a P2P system for wireless devices. Chapter 3, 4 and 5 contain my contribution to this thesis, where I first described how such system could be realized and created a small prototype which was tested it in a small domain. The results indicate that mobile phones can act as leaf Peers, but is still dependent on a fixed P2P network to handle the heavy workload.

In this chapter I will analyze and discuss my findings, the possibilities of the system and limitations I have experienced building the system. The prototype has been subject to different tests, and the results will be analyzed.

I will also discuss how the system complies with resource constraints and loss of connectivity.

6.2 Research discussion

Today's Mobile devices are not confronted by as many constraints as a few years ago and the technology is one of the fastest growing in the industry. The devices are now powerful enough to participate in a P2P network. With memory cards and dynamic heap memory (RAM), a mobile Peer can send and receive large files such as images or music files. The introduction of third generation mobile network (3G) will also enable faster transfer speed at more affordable, hopefully fixed prices.

P2P systems seem to work best when they combine the idea from first generation and second generation P2P. Using many, distributed index servers preserve the advantage of fast resource discovery and decrease network vulnerability against single-point-of-failure.

JXTA is a flexible P2P framework. It is platform, device and language independent. JXME API allows Mobile devices participate as Peers on the JXTA network. Using this technology to build a P2P system with mobile devices has both advantages and disadvantages. The framework enables the developer to concentrate on his application rather than on the core P2P operations. But with ease of work comes lack of flexibility. The JXTA technology can be unnecessarily extensive for specific P2P solutions.

The Mobile P2P system proposed here must use a fixed network to work. This can be the solution for large mobile P2P networks with thousands of Peers to ease workload such as message routing and indexed advertisements on mobile Peers. For smaller solutions, a Proxyless approach would be desirable. This is only possible for JXME for CDC, such as PDA's. A CLDC of JXME Proxyless would be highly interesting, and porting the classes from CDC to CLDC would be a topical thesis assignment to extend this domain.

6.3 P2P System Design

The P2P System design was based on the literature study and experimenting with the JXTA technology. It exploits the advantages of JXTA for J2ME and presents a way to implement different P2P operations. JXTA for J2ME is a work in progress, and not many commercial applications have been developed with this technology yet. A stable version of JXME 2.0 has now been released, and I believe we will see more mobile P2P solutions using JXME in the next few years, even commercially. The main problems using this platform have been the lack of documentation and tutorials available. I believe my thesis can give a flying start for those who want to start developing P2P solutions on JXME; it includes both theoretical background of JXTA, JXME and working examples of P2P operations.

My solution is not application specific. From the design, virtually any P2P solution can be developed, such as mobile multiplayer games, file sharing applications, chat applications or collaboration programs, such as the one I proposed in my prototype. Mobile applications still have considerable constraints compared to a conventional P2P application, but I think these limitations will be minimized within a year or two. Until then, I suggest the mobile Peer should depend on a fixed Proxy Service to provide the core P2P operations.

The disadvantage of a Proxy Service is that the user must provide the IP-address of one or more relay Peers running on a connected computer. The problem I experienced was that the Proxy Service needed constant attention, such as memory flushing to work. This is a considerable problem for the reliability of the system. In my prototype, the Proxy Service needed to flush its advertisement cache before each session.

6.4 The Prototype

The application developed is meant as an example on how to implement the P2P system. The scenario idea was to show how a mobile P2P application can assist in large, chaotic rescue operations to share information in groups of common interest. This can assist rescue personnel to locate where resources are most needed.

Implementing this scenario shows the key potential of a mobile P2P system; the fact that you can bring you mobile everywhere gives a Peer another dimension. Using the Location API for J2ME [36] would enable the application to use positioning coordinates from a GPS or other positioning systems to send its geographical position to other Peers. I chose to simplify this process by letting the user move his Peer icon with the arrow-keys, because I only used mobile emulators to test the system.

Another useful potential I wanted to simulate was the camera function. Using the mobile's camera in the application when a picture is requested is requested would be possible with the Mobile Media API [37], but this proved to be too time demanding to implement and difficult to test. A simple image download function simulated image capturing for my application.

6.5 Test results

6.5.1 Functional requirement testing

The functions of the application work according to its requirement specification. What this test does not show is if the functions work in every possible situation. Since it is just a prototype, all possible exceptions are not caught, and error handling has been set for debugging, and not so much for the users. If this application was to be further developed for real use, non-functional requirements would be more important to analyze and test.

6.5.2 Network testing

Testing the prototype in different network environments show that the application works in larger networks and that the Peers are able to communicate and collaborate even over large distances globally. The performance of the system has not been tested on a large scale, and the amount of messages sent between thousands of Peers may not perform as desired.

JXME Proxy Services are supposed to relieve some of the workload from the JXME Peers. This is also the case in my system, but not to the extent desirable. Advertisements such as Peer info, Pipes and the shared map file should be stored on this proxy, letting all Peers search for resources on the JXME Proxy Service instead of the Peer directly. This is the vision of JXTA, but did not work very well in my application because of the JXTA Shell's shortcomings. If I had known what I now know, I would have implemented the JXTA Proxy Service myself, or altered the JXTA Shell's source code. This would probably have increased the reliability and usability of the system.

6.5.3 Large File Transfer test

This was a considerable task, since JXME does not provide this functionality. Unlike JXTA, JXME does not segment messages, making large file transfer impossible with the API provided. A JXME Message has a maximum size limit of just above 60 kB.

My solution enables message segmentation for large file transfer. This was implemented in the prototype and tested in chapter 3.4.3. The key variable in both of these file transfer tests was the *buffer size*. Analyzing diagram 37 and 38 show that transferring many small segments of data perform much worse than transferring fewer, larger segments. The graph drops very fast between 4 000 and 12 000 bytes, and starts to level at 16 000 bytes. Since transferring large segments to the proxy service are not desirable, choosing the best buffer size would be to find the segment size at the point when the graph starts to level, which is between 16 000 and 20 000 bytes. This is the best segment size considering both *transfer time* and *memory load* on the Peers on the network.

6.6 Other Experiences

6.6.1 Keeping the Peer Connected

Once the Peer is connected to the network, it will not be disconnected if the device temporarily looses connectivity. The Pipe Advertisement on the JXME Proxy Service and his state ID stays as long as the application is running.

The peer only loses the network if the battery runs out or the application crashes for some reason, and must then reconnect as a new Peer. One solution to solve this problem would be to store some key data, such as the Pipe Id and PeerNetwork state information, in the Record Management System (RMS) on the Mobile device. This way it could be possible for the Peer to continue as if nothing had happened. All data waiting to be sent to the Peer is queued at the JXME Proxy Service and sent when the Peer polls for new data.

If a Proxy Service goes down, a Peer should be able to connect to another Proxy Service automatically. This is not implemented, but all that is needed is a list of working Proxy Services available on the JXME Peer. This should improve network reliability considerably.

6.6.2 Advertisements and Peer Discovery

As stated in the literature study on JXTA, advertisements are used to advertise resources such as Peers, Pipes or PeerGroups on the JXTA network. I chose in my design not to use this for resource discovery, other than to search for own Pipes and the PeerGroup. The Peer and resource discovery is supposed to be very easy to handle with advertisements. In theory, one can just search for this on the JXME Proxy Service, thus avoiding unnecessary communication with the Mobile Peers.

The problem I met here was that once you have created an advertisement on the JXME Proxy Service, it could not be removed again by the Mobile Peer. This had to be done manually on the JXTA Shell by flushing its cache memory. This caused problems when resources or Peers were removed from the network; for the other Peers, the resources would still seem available.

The solution was active discovery. Each Peer broadcasted messages when it connected and exited the network and asked the Peer directly for resources. This works fine on a smaller scale, but would cause large overhead traffic on a larger scale.

Again, the problem lies between the JXME Peer and the JXTA Shell. A better Proxy Service implementation would solve this problem and improve the system performance and reliability.

6.7 Summary

Peer-to-Peer programming on wireless devices is an innovative domain with great potential, and commercial implementations will probably be growing similar to the popularity of conventional Peer-to-Peer we have seen in the last few years.

The system I have proposed uses a Proxy Service on a fixed network to ease the workload on constrained mobile phones. With the increasing power of mobile phones, a Proxyless version of JXME would be possible, giving developers the freedom to avoid the fixed JXTA network, enabling pure mobile P2P networks.

Large file sharing is mostly constrained by a mobiles heap memory; my testing of file segmentation shows that the images are segmented and re-assembled correctly as long as the phone has enough available memory.

7 Conclusions and Future Research

7.1 Conclusions

Developing Peer-to-Peer networks for wireless devices is a new and promising research area, rising with the new mobile technology. With this thesis, I have proposed a way to implement a mobile Peer-to-Peer system based on the common protocols of the JXTA technology, an effort to define a common framework for P2P applications. JXTA is a highly complex P2P framework with many protocols and concepts to keep track of. . The JXME API is a lightweight version of JXTA, which lets the developer create mobile P2P solutions fast without extensive knowledge of JXTA.

Because of a mobile device's technological constraints, the best wireless P2P solution should involve more powerful peers acting as a proxy and relays on a fixed network. This gives both the advantages of a fixed P2P network and the mobility of a wireless device.

Since the JXTA for J2ME technology is under development, the documentation and tutorials available are very limited. With this thesis, further development based on this domain should have a solid foundation for understanding the technology and implementing a P2P system.

The system designed can be used to develop virtually any mobile P2P application; mobile multiplayer games, monitoring systems, instant messaging or file sharing to name a few. Such systems are very likely to grow rapidly with the introduction of third generation mobile networks and the ever improving memory and computing power on a mobile phone. I believe mobile P2P technology will be the next big thing in the technology industry; not only does it enable direct search and sharing of multimedia content such as images, mobile games, music and video, but mobile devices has the extra dimension of being a personal device that is with us all the time, and in many ways defines us as individuals. The challenge will be to take advantage of this fact and find new areas to utilize the technology.

7.2 Future research

To improve the mobile P2P system, one should look more closely at the JXME Proxy Service to cope with the problems I have encountered. This would improve the system functionality and utilize the Proxy Service much more than the proposed system does.

The prototype has made a lot of simplifications, and it would be interesting to see how it performs on real mobile devices. It would be especially interesting to implement the camera-function to share images taken with the mobile's camera and the Location-function to distribute geographical data automatically when the Peer is moving.

Even though the system proposed is working, I believe a Proxyless solution would be more successful in smaller P2P solution and is the right way to go for further research and development of pure wireless Peer-to-Peer programming. I propose developers who want to continue this research to join the JXTA community effort to port the JXME Proxyless implementation from the CDC environment to the CLDC environment.

References

[1] Gong, Li. *Project JXTA: A Technology Overview*, Sun Microsystems, Inc, 2002

[2] Biström, Johnny and Partanen, Ville. *Mobile P2P - Creating a mobile file-sharing environment,* Helsinki University of Technology, 2004.

[3] JXTA Project,
[web site] http://www.jxta.org

[4] Dochstader, Mark. *Peer to Peer Networking: Next Big Boom or Next Big Bust?,* ITtoolbox Networking, 2001-03-12

[5] Galla, Preson. *Five Live Ones*, Darwin Magazine,
[online] http://www.darwinmag.com, 2001-08-01

[6] [accessed] 2005-05-25

[7] CacheLogic.
[web site] http://www.cachelogic.com.
[accessed] 2005-04-06

[8] Sun Microsystems, Java 2 Platform, Micro Edition,
[online] http://www.java.sun.com/j2me/docs,
[accessed] 2005-06-14

[9] JXME project,
[web site] http://jxme.jxta.org

[10] Arora, Akhil, Haywood, Carl and Kuldip Singh Pabla, *JXTA for J2ME™ – Extending the Reach of Wireless with JXTA Technology*. Sun Microsystems Inc., March 2002

[11] TINI Project,
[web site] http://tini.jxta.org

[12] Unknown. *Differences between PersonalJava and MIDP Java Environments*, Symbian,
[online]
www.symbian.com/developer/techlib/papers/PJAE_MIDP/PJAE_MIDP_2.pdf,
[accessed]14-06-2005

[13] Wilson, Brendon. *JXTA*, New Riders,
[online] http://www.brendonwilson.com/projects/jxta,
[accessed] 2005-06-14

[14] Traversat, Bernard and Arora, Ahkil. *Project JXTA 2.0 Super-Peer Virtual Network*, Sun Microsystems, Inc, 25-05-2003

[15] Ortiz, Enrique. *A Survey of J2ME Today*, Sun Microsystems, October 2004

[16] EmbeddedJava technology,
[online] http://www.java.sun.com/products/embeddedjava/, Sun Microsystems,
[accessed] 2005-05-06

[17] PersonalJava technology,
[online] http://www.java.sun.com/products/personaljava/, Sun Microsystems,
[accessed] 2005-05-06

[18] Enrique Ortiz, *An Introduction to Java Card Technology - Part 1*, Sun Microsystems, 2003-05-29,
[online]
http://developers.sun.com/techtopics/mobility/javacard/articles/javacard1

[19] *JSR-36 J2ME Connected Device Configuration 1.0.1*
[online] http://jcp.org/aboutJava/communityprocess/final/jsr36/

[20] *JSR-218 J2ME Connected Device Configuration 1.1*
[online] http://jcp.org/aboutJava/communityprocess/final/jsr218/

[21] Connected Device Configuration (CDC),
[online] http://www.java.sun.com/products/cdc/, Sun Microsystems,
[accessed] 2005-05-06

[22] *JSR-30 J2ME Connected Limited Device Configuration 1.0*
[online] http://jcp.org/aboutJava/communityprocess/final/jsr30/

[23] *JSR-139 J2ME Connected Limited Device Configuration 1.1*
[online] http://jcp.org/aboutJava/communityprocess/final/jsr139/

[24] Connected Device Configuration (CDC),
[online] http://www.java.sun.com/products/cdc/, Sun Microsystems,
[accessed] 2005-05-06

[25] Antoniu, Gabriel, Hatcher, Phil, Jan, Mathieu and Noblet ,David A. *Performance Evaluation of JXTA Communication Layers*, University of New Hampshire Department of Computer Science Durham, New Hampshire, 2005

[26] Wireless Tool Kit,
[web site] http://www. java.sun.com/products/j2mewtoolkit

[27] Sony Ericsson,
[web site] http://www.sonyericsson.com

[28] *JSR-37 J2ME Mobile Information Device profile*
[online] http://jcp.org/aboutJava/communityprocess/final/jsr37/

[29] *JSR-201 J2ME Information Module profile*
[online] http://jcp.org/aboutJava/communityprocess/final/jsr201/

[30] *JSR-68 J2ME Platform Specification*
[online] http://jcp.org/aboutJava/communityprocess/final/jsr68/

[31] *JSR-118 J2ME Mobile Information Device profile 2.0*
[online] http://jcp.org/aboutJava/communityprocess/final/jsr118/

[32] *Wikipedia, the free encyclopaedia*,
[web site] http://wikipedia.org

[33] Jonathan Knudsen, *Understanding MIDlet Memory*, Sun Developer Network, 2002-06-07,
[online] http://developers.sun.com/techtopics/mobility/midp/ttips/memory,
[accessed] 2005-06-13

[34] Benhui.net, MIDP 2.0 Phone Resources,
[online] http://www.benhui.net/modules.php?name=Midp2Phones,
[accessed] 2005-06-13

[35] *JSR-75 File Connection API*
[online] http://jcp.org/aboutJava/communityprocess/final/jsr75/

[36] *JSR-179 Location API for J2ME*
[online] http://jcp.org/aboutJava/communityprocess/final/jsr179/

[37] *JSR-179 Mobile Media API*
[online] http://jcp.org/aboutJava/communityprocess/final/jsr135/