Network technologies for Java-enabled, mobile devices

Master thesis

Bjørg Peggy Sveen Lyngstad

Trondheim, July 2002

Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Telematics

Preface

This report is the result of my master thesis, and concludes my years as student at NTNU. The master thesis was suggested by myself as a continuation of my project assigment that was carried out during the autumn 2001.

I would like to give my thanks to everyone that have supported and assisted me during the past half-year, especially those who shared their thoughts around the subjects. A special thanks must be given to Jorunn Kaasin for a valuable and creative co-operation in the development of the prototype, which is part of both our master thesis'.

I would also give a special thanks to my supervisors, Ulrik Johansen and Ståle Walderhaug at SINTEF Telecom and Informatics for valuable comments on my work, to professor Rolv Bræk for advices, and to Joakim for reading and commenting the report.

Trondheim, 8th July 2002

Bjørg Peggy Sveen Lyngstad

Abstract

Computers are changing radically, size decrease and they become increasingly mobile. As a result of the decrease in size, the computers have limitations in capacity of processing and available memory. The evolution moves computers from being permanently attached, to become mobile devices, and this mobility results in a change from hard-wired network connection to wireless connection.

The new embedded devices that are highly mobile creates a demand for network technologies that makes it able to have access to services anywhere, at any time, and on any device. An client/server architecture is not the best solution is such highly mobile and heterogeneous environment, neither is the older distributed network technologies like CORBA and DCOM. A more suitable solution is to use new distributed peer-to-peer network technologies, with its independence on infrastructure and able to handle a highly mobile environment where nodes come and goes with no further notice. The challenge is to make these technologies work with the application environment on the devices.

Java 2 Micro Edition (J2ME), the new Java programming environment for embedded devices, has several capabilities when it comes to network connections. Concentrating on the Connected, Limited Device Configuration (CLDC) and Mobile Information Device Configuration (MIDP), these specifications provide network connection through the Generic Connection Framework defined in CLDC. MIDP only implements the HTTP1.1 protocol, and the only possibilitiy to use more advanced network protocols are by use of a "middleman", or relay. The relay can act on the MIDP device's behalf, and can provide access to for example CORBA and DCOM, or to the new peer-to-peer network technologies Jini and JXTA.

Jini is a distributed network technology that is designed to handle dynamic environments. It is based on the assumption that Java is present in the network, and ensures a homogeneous environment by use of mobile code that is transferred between clients in the network. All resources in the network are defined as services, and can be found through a lookup service, a sort of repository holding reference to all services. Jini provides the surrogate architecture to make resource constrained and/or non-java enabled devices able to participate in the Jini community.

JXTA is a distributed platform that tries to standardize peer-to-peer. The objective is to provide interoperability between entities in the network, be platform independent and offer ubiquity, i.e. any device with a digital heartbeat can participate. JXTA defines a number of concepts that are common for peer-to-peer networks like peer, peer group and pipes. These concepts are the primary components of the JXTA platform. The base of the JXTA specification is a set of protocols that enable discovery of resources on the network and the ability to connect and communicate with these resources. JXTA provides a package that makes it possible for J2ME-enabled devices to participate in the community.

To find out the possibilities and limitations of JXTA for J2ME, a prototype was developed as part of the master thesis. The prototype, called Intelligent Traffic Service System (ITSSystem) was developed in cooperation with Jorunn Kaasin, and the essential feature is the ability for users to communicate with each other dependent upon their location. It includes requirements specification, design, implementation

notes and test specifications and result. The prototype showed that JXTA for J2ME is easy to use, and has the most necessary capabilities, but the lack of more advanced capabilities limits the possibilities.

Several other initiatives than Jini and JXTA have been made to provide distributed platforms that are able to communicate between different entitites reagardless of platform and implementation language. They all differs in the way the independence is provided. Web Services is a service architecture based on usage of XML, SOAP, WDSL and UDDI. Microsoft .NET is Microsoft's answer to distributed platforms. The infrastructure, the .NET Framework, provides runtime services and class libraries, and .NET services will always run where .NET is installed.

Jini and JXTA might provide capabilities that are valuable in relation to existing development project. AMIGOS (Advanced Multimedia In Group Organized Services), a project at Department of Telematics, NTNU, is a service that provides users with the possibility to establish and participate in Meeting Places, i.e. virtual rooms where interaction optionally takes place between participants. A Meeting place might be limited to geographical areas, and interaction includes exchange of information and media streams like text, audio and graphics. Use of Jini and JXTA in realtion to this project will give a totally different solution than the one envisioned by the developers. The AMIGOS architecture of today is very centralized, but with introduction of Jini or JXTA it would become distributed and the functionality of AMIGOS might be split in several parts.

Table Of Contents

List Of Figures

List Of Tables

Chapter 1

Introduction

"The greatest challenge to any thinker is stating the problem in a way that will allow a solution."
- Bertrand Russel

Computers are changing radically. Processing power and memory capacity continue to double every 18 months according to Moore's Law, and the size decreases. Scientists predict that within the next few years, the most prevalent computers will be small mobile wireless devices such as cell phones, personal digital assistants (PDA), and pocket-sized PCs. IDC, a market research firm, projects that the worldwide annual shipment of intelligent handheld devices will grow from about 20 million in 2001 to about 62 million by 2004 [Kiely].

The evolution moves computers from being permanently attached to being mobile devices, and this mobility results in a change from hard-wired network connections to wireless connections. Any randomly assembled collection of small, mobile wireless devices can be expected to present a highly heterogeneous computing environment. This comes as a result of the fact that there is no one manufacturer that dominates the operating system marked for embedded devices. They use a variety of special purpose operating systems and processors, and are programmed in different languages. Further, mobile devices is constantly on the move, coming into and going out of contact with new devices as their users move around.

1.1 The assignment

Today, mobile, wireless devices are used as an extension to the existing wired network and thought of as just additional client devices with wireless connection. For example, cell phones can be used to send and receive email and instant messaging, SMS. But the devices have an unfulfilled potential: to form personal area networks (PANs) and participate in distributed applications. New technologies will soon make this scenario possible.

Applications for mobile, wireless devices must overcome the diversity in platforms and operating systems. To get an application running on all devices, it must platform independent. This can be achieved by use of the Java Programming Language. Java is not just a language, but also a software platform, i.e. programming environment, that has gained great success after its introduction in 1995. The basic characteristic of Java is that it hides the complexity of devices from applications. Applications see the standardized interfaces of Java, and do not have to deal with the special characteristics of different devices, and as a result applications are portable between different platforms. Java 2 Micro Edition is a version of Java designed to run on small devices with limited memory, and several cell phones and PDAs with J2ME support are available with more on the way.

Low-level technology needed to make ad hoc peer-to-peer mobile distributed architectures a reality is already available, like WLAN and Bluetooth. High-level network technologies are under development: two initiatives from Sun, JXTA and Jini, both support applications in a mobile environment.

The master thesis will be based on a project assignment that was carried out in autumn 2001, "Java on Mobile Devices", and will not include aspects that were covered in that report. The assignment is to investigate the networking capabilities in the J2ME version for mobile devices like cell phones and PDAs, and find explore abilities and limitations of the network technologies Jini and JXTA.

This report will give a short introduction to J2ME and CLDC/MIDP, Java APIs for embedded devices like mobile phones, and then focus on the network abilities provided as part of these platforms. Two distributed network technologies, Jini and JXTA will be introduces and an overview over both technologies will be provided. Both Jini and JXTA have solutions for incorporation of embedded, mobile devices; the surrogate architecture and the JXTA for J2ME, respectively, which both will be presented.

The use of Jini and JXTA will be discussed to find advantages and limitations, and possible usage of the two technologies in the AMIGOS project at Department of Telematics, NTNU, should be presented.

A prototype is to be developed as part of the master thesis. One of the requirements is that it should be carried out as in a software development project, including requirement specification, design, implementation and test. Depending upon study of Jini and JXTA, the most suitable of the two technologies should be used in the prototype.

1.2 Objectives and limitations

The objective of the assignment is to get an overview of the capabilities Java-enabled devices have to communicate in networks. One of the critical aspects of J2ME is network connectivity, but the question is how the abilities provided today can be used and possible extended to support features outside the ordinary client-server architecture.

Technologies related to client-server processing, and the support of different types of terminals accessing a server will not be considered or discussed. Neither will low-level peer-to-peer technologies like Bluetooth and WLAN, nor access technologies like GSM, GPRS and UMTS. Other areas that will not be covered in relation to J2ME are technologies like XML, SOAP or SyncML. XML is a markup language that handles structured information, SOAP is a XML/HTTP-based protocol for accessing services, objects and servers in a platform independent way, while SyncML is used to synchronize data over a network.

1.3 Reader's guide

As this report is extensive, and some parts might be of more interest for some readers than others, this reader's guide is included to guide the readers in finding what they are looking for. First a presentation of the chapters will be given, and then some tips for different kinds of readers.

- **Chapter 1 Introduction**: The chapter you are reading now. Introduces the assignment, its objectives and limitations. In addition it provides a reader's guide.
- **Chapter 2 Background**: The document starts with a background chapter that gives and introduction to mobile, embedded devices and the network technology paradigms client/server and peer-to-peer.
- **Chapter 3 Network capabilities in J2ME**: Presents the J2ME with focus on CLDC and MIDP, and the network capabilities in these technologies today and what capabilities are on their way.
- **Chapter 4 Jini Network Technology**: Presents the Jini Network Technology and the surrogate architecture for embedded devices.

- **Chapter 5 JXTA**: Focus on JXTA, a peer-to-peer architecture from Sun, and the JXTA for J2ME package aimed at J2ME-enabled devices.
- **Chapter 6 Related work**: The chapter is dedicated to related work, like the AMIGOS project that might find Jini and JXTA useful, and the competitive technologies Web Services and Microsoft .NET.
- **Chapter 7 The Prototype**: Describes the prototype that was developed in relation to the project in cooperation with Jorunn Kaasin. The prototype is called Intelligent Traffic Service System (ITSSystem). Javadoc and Java code is enclosed on the CD accompanying the report.
- **Chapter 8 Discussion**: The discussion includes comparison of JXTA and Jini, and a discussion around their solution for embedded, mobile devices. A melting of the two network technologies is considered, and how they differ from .NET and Web Services. The chapter also includes a discussion around future use of Jini and JXTA in the AMIGOS project, and a discussion of the prototype.
- **Chapter 9 Conclusion**: The report is concluded in chapter 9 with some recommendation and thoughts around the future of the technologies discussed.

The report also includes a number of appendixes in relation to the subject presented in different chapters. These are not considered directly connected to the assignment, but just as additional information for the interested reader.

Parts of the chapters or whole chapters can be read if the reader is interested, but they are not mandatory to understand the discussion and conclusion. Chapter 2 to 6 are independet of the other chapters, while chapter 7 require some knowledge of J2ME MIDP, which is covered in chapter 3, and JXTA for J2ME, covered in chapter 5. The different sections in the discussion in Chapter 8 is dependent upon different chapters, and these dependencies are summarized in the table below.

Section of chapter 8	Depends upon chapter
8.1	4, 5
8.2	4.4, 5.6
8.3	4, 5
8.4	4, 5, 6.2, 6.3
8.5	2, 3, 5, 7
8.6	4, 5, 6.1
8.7	-

Table 1-1 Dependencies in chapter 8

There are different ways to read this report, according to time and interest. Each chapter includes a summary at the end where the most important points in the chapter are summarized. A reader with limited time should concentrate on the abstract, the summaries in each chapter and the conclusion. For a more in-depth understanding of the conclusion, a reader can supply the above-mentioned parts with the discussion in chapter 8.

Jini and JXTA in AMIGOS: A reader interested in the use of Jini and JXTA in the AMIGOS project should read chapter 4 and chapter 5, and the part of the discussion concerned with AMIGOS. If the reader is not familiar with AMIGOS, the part concerning AMIGOS in chapter 6 should be included.

Jini and JXTA: A user interested in Jini and JXTA and how they compare to related technologies should concentrate on chapter 4 and chapter 5, in addition to the parts in chapter 6 presenting Web Services and Microsoft .NET. In addition the reader should read the brief discussion in chapter 8.

JXTA: If JXTA is the area of interest, and the usage of the technology in relation to J2ME MIDP, the reader should concentrate on chapter 3, chapter 5 and the prototype in chapter 7. In addition the reader should take a look at the parts of chapter 8 that discusses the prototype.

Software development project: Readers interested in the software development project should concentrate on chapter 7, and the part of chapter 8 that discusses the prototype. Several of the appendixs also includes information about the prototype like appendix C, D, E and F.

Chapter 2

Background

"Discovery consists in seeing what everyone else has seen and think what no one else has thought."
Albert Szent-Gyorgi

Embedded devices have gained great success, and analytics recognise that these computers will become a common part of our society during the next decades. The technology has many possibilities, but also limitations. Both will influence the networking capabilities for embedded devices.

This chapter will start by providing an overview of embedded devices, their capabilities and limitations in general and in relation to application development and network technologies. It then moves on to give an introduction to network technologies, before moving on to a presentation of the client/server network architecture and the new peer-to-peer network architecture.

2.1 Embedded devices

Mobile embedded devices are small, microprocessor-based consumer products, like hand-held battery-operated products such as cell phones, two-way pagers, and personal portable organizers. They where named embedded devices because the small computers inside them have a very focused operation. Most mobile devices communicate with other devices by a wireless link.

When implementing applications for embedded, mobile devices, their limitations have to be taken into account. These limitations are:

- Limited memory and processing power
- Limitations in input possibilities, normally a keyboard with approximately 12-18 keys
- Small screens
- Limited network connection possibilities
- Battery driven with limited lifetime

Applications must also take security issues into consideration. During the last years, several initiatives have been started in order to develop an application development environment for embedded devices. Most of these have been aimed at the Palm Platform, like IBM and their Java virtual machine J9 and the VAME (Visual Age Micro Edition) IDE that wrap all native C methods for the Palm API in Java. Another example is the Waba [Waba] or SuperWaba [SWaba] solution.

Java 2 Micro Edition (J2ME), a general Java application environment for embedded devices, came as an initiative from Sun in 1999. The platform provides functionality that suites devices with different demands within the embedded market. J2ME has gained much attention the last year, and many companies, like Siemens, Motorola and Nokia, seem to adopt the platform in their phones. An overview of J2ME is presented in chapter 3.

Mobile applications also have several advantages in contrast to ordinary applications, as we know it today. They are able to be highly mobile, services can be dependent on location and a mobile application makes it possible for the user to always be connected. But there exist some constraints that have to be taken into account when designing distributed, mobile, network applications.

Embedded, mobile devices, like mobile phones and PDAs operate on the edge of the network, and as a result of their mobile nature, they require network technologies that are able to operate in an environment where some or all the nodes are mobile. Today, the most common wireless technologies are GSM, GPRS and WLAN. Soon, other technologies like UMTS and Bluetooth will be available. These technologies are mostly used as access-technologies to the Internet, but the hope for the future is for the devices to form network without dependence on Internet connection. To be able to create such an environment, the network functionality must run in a distributed fashion. Here, nodes will enter and leave the network without further notice, while the users will have the same demands for connectivity and traffic delivery as in traditional networks.

Typical characteristics for network functions, when concentrating on networks of computers (mobile devices and PDAs), are:

- Distributed operation.
- Dynamic network topology.
- Variable link capacity; i.e. less connection stability and variety in connection speed, as well as varying network coverage.
- Devices with limited capacity.
- Possible high latency connection.

Since embedded devices have limited capacity, like a small processor, small amount of memory and battery driven, this has to be taken into account when the device becomes part of a network. As a result, all algorithms and mechanisms implementing the network functionality must be optimized for small power usage. Applications written for embedded devices to operate over a network should also be able to adapt to sudden changes in the transmission quality. The reason is the variable communication quality in wireless networks, which makes it difficult to guarantee for the services that is advertised for the device.

2.2 Network architecture and technologies

Developing applications executing locally on an embedded device may be useful enough for some, but most users demand connectivity to a network. They want to be able to have access to services anywhere, at any time, regardless of device.

Traditional network technology has some limitations that puts restrictions on applications and reduces the applicability. To compensate for these drawbacks and allow for sharing of applications over a network, distributed technologies like CORBA, TINA and DCOM was developed. The newest leaf to distributed network technology paradigm tree is the *Peer-to-Peer (P2P) network technology*.

2.2.1 Client/Server Architecture

In the client/server network architecture, the client connects to a server using a specific communication protocol to obtain access to a specific resource. One such protocol could be Hypertext Transfer Protocol (HTTP) or File Transfer Protocol (FTP). In this scenario, the burden is places upon the server when it comes to processing and delivery of the service, and the requirements for the client is small. This architecture is used by most popular Internet applications, like the Web and e-mail. The client/server architecture is shown in Figure 2-1.

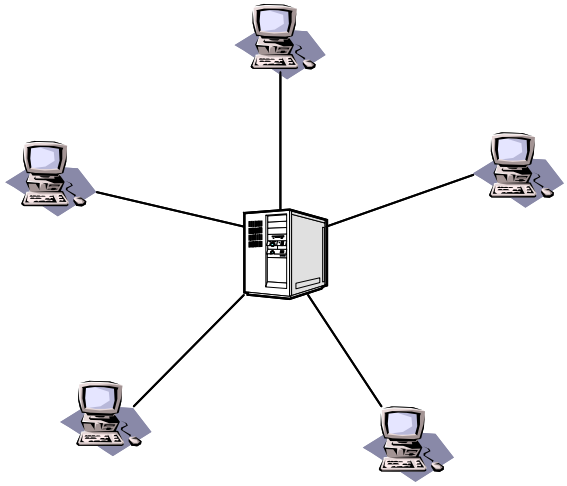


Figure 2-1 Client/Server Architecture

The background for adopting this client/server strategy from the beginning was the availability of computational power. Equipment was expensive and one could not expect the client side to possess much computational power. On the other hand this model was developed at a time when most machines on the Internet had resolvable static IP addresses, it was possible to find a machine using a simple name that was translated to an IP-address by use of a DNS.

Today we can observe that this architecture has several advantages and some major drawback. The advantages are:

- **Simple**: The primary advantage of centralized systems is their simplicity.
- **Manageability and information coherence**: All data is concentrated in one place, and as a result they are easy to manage and have no question of data consistency or coherence, i.e. if a bit of data is found in the system, the data is correct.
- **Security**: Centralized systems are relatively easy to secure since there is only one host that needs to be protected.

Major drawbacks are:

- **Scalability**: Theoretically, since the number of clients increase, the load and bandwidth demands on the server also increase. This would eventually prevent the server from handling additional clients. In practice, a server may have enough resources to server the demands of its users as a result of the growing CPU and memory capacity. [Minar]
- **Bandwidth**: The bandwidth capacity has increased by a factor of 10^6 since 1975, all according to Moore's law. But most of this bandwidth is not used if everyone just goes to the same few sites to get their services.
- **Computational power**: Many people have invested in computational power that are oversize the processing power needed to use the most popular Internet applications like surfing the web and reading/sending e-mail. In other words, there might be a lot of unused processing capacity in the client today. This suggests that the client-side could do more of the processing than they do in client/server architectures.
- **Fault tolerance**: The network depends on central points, namely server, to provide service. If the central server goes down, everything does.
- **Extensibility**: Centralized systems are hard to extend since resources can only be added to the central system.

2.2.2 Peer-to-Peer Network Architecture

Distributed network technologies try to compensate for some of the limitations of the client/server architecture. They provide a client with an interface to call for services, and so limits the client's knowledge of where to find the services. The services are distributed among a number of servers, and the client's request is sent to one, or several, of those, but the client is unaware of who is responding. But this architecture classifies computers in servers and clients, and only servers can provide an answer to a request. The earlier distributed network technologies like CORBA and DCOM are highly advanced, and are to extensive and heavy for embedded devices. Later distributed network technologies have included the mobile environment at design time, and hence is also available for the mobile devices.

An architecture that has gained much attention the last years is the peer-to-peer architecture. Figure 2-2 shows the Peer-to-Peer architecture, consisting of interconnected nodes. The P2P paradigm consists of distributed network technologies like Jini and JXTA, discussed in later chapter of this report. Peer-to-peer came into existence as a result of different possibilities:

- **Decentralizing**: It is a natural result of decentralizing trends in software engineering intersecting with available technology.
- **Powerful networked computers and inexpensive bandwidth**: Intersecting this trend is the growth in the available of powerful networked computers and inexpensive bandwidth. Peer-to-peer requires the availability of numerous, interconnected peers to be effective.
- **Popular programs**: Also non-technical issues was important, like the popularity of products like Napster, Gnutella and similar programs.

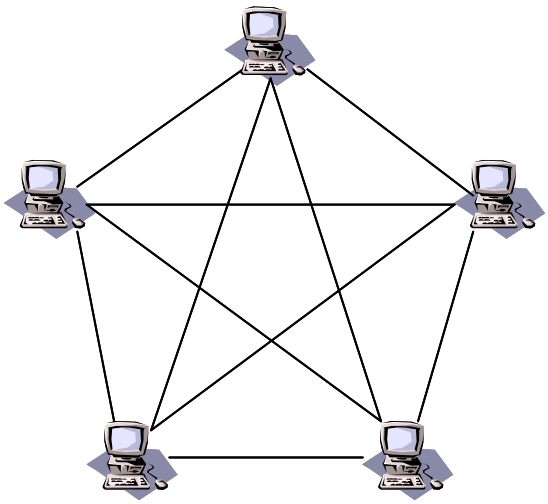


Figure 2-2 Peer-to-peer Architecture

Decentralized architectures like peer-to-peer have almost opposite characteristics as centralized systems. The favourable properties of the peer-to-peer architecture are:

- **Capacity**: Takes advantages of unused processing capacity in the Internet. Exploit available bandwidth across the entire network by using a variety of communication channels, and filling bandwidth to the "edge" of the Internet.
- **Independence**: Prevent the dependency on a central server to provide services
- **Configuration**: Remove the limitations of demand for configuration before use and is therefore classified as a self-configured network architecture.
- **Decentralized**: Consists of mobile nodes where each node has the same status and can request and provide services for each other, and create a network without a central control. Reduce network congestion by enable communication via a variety of network routes.

- **Extensibility**: Any node can join the network and instantly make new resources available to the whole network.
- **Fault tolerance**: Failure or shutdown of any particular node does not impact the rest of the system, i.e. a service is not unreachable due to a single point of failure.
- **Scalability**: In theory, the more hosts that are added to a decentralized system, the more capable a decentralized network becomes.

The drawbacks of decentralized architectures are:

- **Manageability and information coherence**: Distributed networks are widespread, and as a result they are difficult to manage. In addition the data in the system is never fully authoritative, i.e. a bit of data that is found cannot be trusted to be correct.
- **Security**: The networks tend to be insecure in the sense that it is easy for a node to join the network and start putting bad data into the system.
- **Scalability**: Scalability was listed as an advantage, but only in theory. In practice, the algorithms that are required to keep a decentralized system coherent often carry a lot of overhead. The system may not scale well if the overhead grows with the size of the system. [Minar]

Peer-to-peer is often mixed with ad-hoc networks (see appendix B), and many claims it is the same phenomenon. One may say that peer-to-peer is a paradigm, and ad-hoc is a realisation of the paradigm where the peer-to-peer functionality is placed in the lower layers of the protocol stack and therefore is dependent on the physical medium. Other peer-to-peer realisations might concentrate on making the implementation of the paradigm independent on the network technology, and so solves the problems at the application layer. The term peer-to-peer is mostly used about the latter.

There are two major forms for the peer-to-peer paradigm:

- **Pure**: All nodes are Peers, and each Peer may function as router, client, or server, according to the status of the query. This form is presented above.
- **Hybrid**: Some nodes are router-terminals that facilitate the interconnections between peers.

Hybrid Peer-to-Peer Architecture

The notion of peer-to-peer has been extended to cover a range of protocols and solution that does not fully satisfy the pure peer-to-peer definition. Many peer-to-peer protocols have introduced a central element in the peer structure to be able to offer a consistent connection. The central element may be introduced as a static routing table, or as a dynamic combined group manager and routing table. [SkaKau]

To be consistent with the idea of peer-to-peer networking, the management functions should be assigned to the peers. A central routing table with no further functionality than routing is accepted as a necessary element in hybrid peer-to-peer networks.

The central router may be configured to play different roles. One of two typical configurations [Graham] is router as a router in a traditional sense (e.g. Internet routers), while the other solution is more of a look up service (e.g. DNS in Internet). The most peer-to-peer like solution is displayed in model 2 in Figure 2-3.

In this scenario the router does not forward messages as a router in the traditional sense, but contains an address resolution table to assist peers in finding fellow peers. All communication is between the two peers, not through the central 'server', and so the server is downgraded to a simple lookup service. As a result of this, congestion problems are avoided through the relatively small amount of processing when performing lookup. A possibility is to distribute this address resolution table into a distributed catalogue, and hence be able to define the system as pure peer-to-peer protocol. The reoccurring problem will however be; how to gather the sufficient pre-knowledge to register and communicate with other peers, and

at the same time be guaranteed reliable communications?

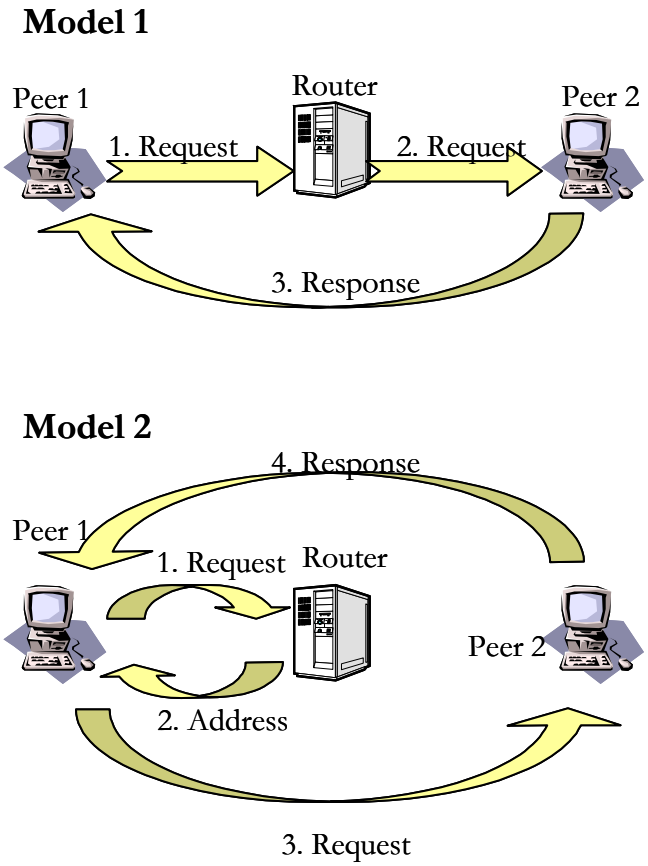


Figure 2-3 Routers in Hybrid Peer-to-Peer networks

A system combining centralized and decentralized elements enjoys some of the advantages of both. Extensibility and fault tolerance is the contribution from decentralization, while the centralisation contribute with more coherence than a purely distributed system since there are fewer hosts that are holding authoritative data. Security and manageability is as difficult as in decentralized systems, but when it comes to scalability it has shown to scale nicely. That is why this architecture is considered a good architecture for peer-to-peer. [Minar]

2.2.3 Where to use peer-to-peer technology?

The most obvious answere to why peer-to-peer technology should be used is to utilize the resources in the network sufficiently. Especially embedded devices will gain much by using the technology, since they can form their own network independent on the Internet architecture's limitations. In areas where no infrastructure exists, ad-hoc networks might be used in rescue operations and for people to communicate from places where there exist other terminals, but no infrastructure.

Peer-to-peer could replace search sites like Google, and provide people out looking for information with up-to-date information. In contrast to search sites today that update their databases once a day, or once a week, each peer would be responsible to return documents and other matches to the search right away.

Another area is in distributed computing. This is a way of solving difficult problems by splitting the problem into sub-problems that can be solved independently by a large number of computers. Until now, the most popular applications of distributed computing have not been peer-to-peer solutions. Examples of

projects that has gained much attention is SETI@home [SETI] that among other assignments analyses data from the universe and distribute the data among computers in screensavers to see if there is life beyond our solar system.

2.3 Existing Peer-to-Peer (P2P) applications

As with all these new concepts, a range of projects has been started to provide the necessary technology to realise the concepts. This includes projects like Napster, ICQ and Gnutella. As most peer-to-peer network applications they are not purely peer-to-peer, but have a hybrid peer-to-peer architecture. A short overview of these projects, which are some of the most well-known peer-to-peer applications, will be presented in this section.

2.3.1 Napster

The Napster [Napster] protocol is composed by clients and servers, and seems in the first place to be nothing like a peer-to-peer networking application. The reason why it Napster is introduced to be the originator of the peer-to-peer paradigm is that it is the first service that take advantage of the possibly enormous amounts of free storage placed in the Internet clients.

One of the problems of using this unused capacity was that clients did not have a static IP address as a result of the ISP's dynamic IP address assignment. This made DNS lookup impossible for address resolution. The solution was to introduce a proprietary protocol that made it possible to bypass the DNS problem, and update the dynamic client IP addresses at real time. This was first introduced by ICQ in 1996. The Napster protocol works on top of existing Internet, and utilizes a proprietary naming service, linking the dynamic client user addresses to a specific Napster name. To be able to maintain the specific Napster naming technique, the Napster server has to be included in every network transaction. The Napster server offers a naming server, a search engine (for mp3 files), and the Napster client application using the services offered by the server.

2.3.2 ICQ

ICQ is an application where users can keep track of their contacts, send instant messages, and share files across the network. It is the first application that made creating a public network address effortless since ICQ does not depend upon knowing IP addresses, domain name servers or hosting facilities. Just give the PC a network address, and that PC can talk to any other PC with an address in the ICQ name space.

2.3.3 Gnutella

Gnutella [Gnutella] is a program that offers sharing, searching and downloading of a large amount of filetypes. Unlike Napster, the Gnutella protocol does not maintain any form of central caches and does not offer a new naming policy to deal with the dynamic client IP addresses. A central IP address table is not necessary either, and the reason why is that the peer IP address is broadcasted when a peer connects to the Gnutella network. Gnutella peers are also connected to different peers for every single connection that is initiated. Even though Gnutella does not provide a central server, it has the same configuration problem as Napster: who to contact to get up and running.

To be able to start communication, Gnutella introduce the concept of "rendezvous points". One such point has to be known to the peers, and these can be found at some Gnutella enthusiast server running gnuCache, which has some similarity with the DHCP functionality.

Some central elements had to be added to make the Gnutella protocol work smoothly, but besides that, the Gnutella is strongly based on peer sharing storage resources. As a result, Gnutella have a hybrid peer-to-peer architecture.

2.4 Summary

Embedded devices have limited resource, and require network technologies that can handle mobility, i.e. that the devices appear and disappear from the network without notice. Applications for embedded devices should be connected to a network, to be able to fulfil the wishes of its users; access to services anywhere, at any time, on any device. Java is an application development platform that looks promising for embedded devices.

Traditional network technology like the client/server architecture puts restrictions on applications and reduces the applicability. To compensate for this, several distributed network technologies were developed, like CORBA and DCOM, but none considered the requirements of the mobile environment before the peer-to-peer paradigm was introduced in the latest distributed network technologies like JXTA and Jini.

Peer-to-Peer is a network technology that does not rely on a central server to provide services, instead the services are distributed among the participating nodes, and the nodes act as client, router, or server depending upon the query it receives. Hybrid Peer-to-Peer architectures are dependent upon a central server to provide access to services.

Several applications have been developed that uses peer-to-peer network architecture, for example Napster, ICQ and Gnutella.

Chapter 3

Network capabilities in J2ME

"You see things; and you say, "Why?"
But I dream things that never were;
and I say, "Why not?" "
George Bernard Shaw
Back to Methuselah

Java 2 Micro Edition (J2ME) is the latest contribution to Sun's Java Platform that in addition consist of Java 2 Standard Edition (J2SE) and Java 2 Enterprise Edition (J2EE). It is a collection of APIs focusing on consumer and embedded devices, ranging from telematic systems, to mobile phones and Personal Digital Assistants (PDAs).

This chapter gives an introduction to the Java 2 Platform, Micro Edition (J2ME) with focus on the Connected, Limited Device Configuration (CLDC) and Mobile Information Device Profile (MIDP). It is meant as background information to the master thesis, as the possibilities and limitations of CLDC and MIDP was covered in a project assignment last autumn [Lyng2001].

The network abilities of CLDC and MIDP are the only area that will be covered in detail, as this is of interest to this master thesis. Even though these abilities are limited, there exist solutions on how to overcome them, like the middleman architecture. Next generation of MIDP will provide further capabilities, and additional profiles are under development at Java Community Process ([JCP], [Lyng2001]).

3.1 Java 2 Micro Edition (J2ME)

The history of J2ME reaches back to Sun's lab where they managed to create a virtual machine implementation on a Palm Pilot. This project was called the Spotless Project [Spotless], and the virtual machine created was moved into the K Virtual Machine (KVM). This virtual machine together with the Spotlet demo was released for early access at JavaOne 1999. The project moved into standardization via the Java Community Process (JCP), and came out as a set of configurations and profiles, and the new name Java 2 Micro Edition.

Sun had made a couple of attempts to make java smaller prior to the introduction of the K virtual machine (KVM). Those where the Java Card, Embedded Java and Personal Java. J2ME will replace Embedded Java [EJava], while Personal Java [PJava] is under standardization to be included as a profile in J2ME called Personal Profile [J2MEFAQ]. Java Card [JCard] aims at running Java programs on smart cards, i.e. has different goals than J2ME, and will exist as a technology on its own.

J2ME has a software layer stack consisting of three layers on top of the Host Operation System of the device. These are the Java Virtual Machine Layer, Configuration Layer and Profile Layer.

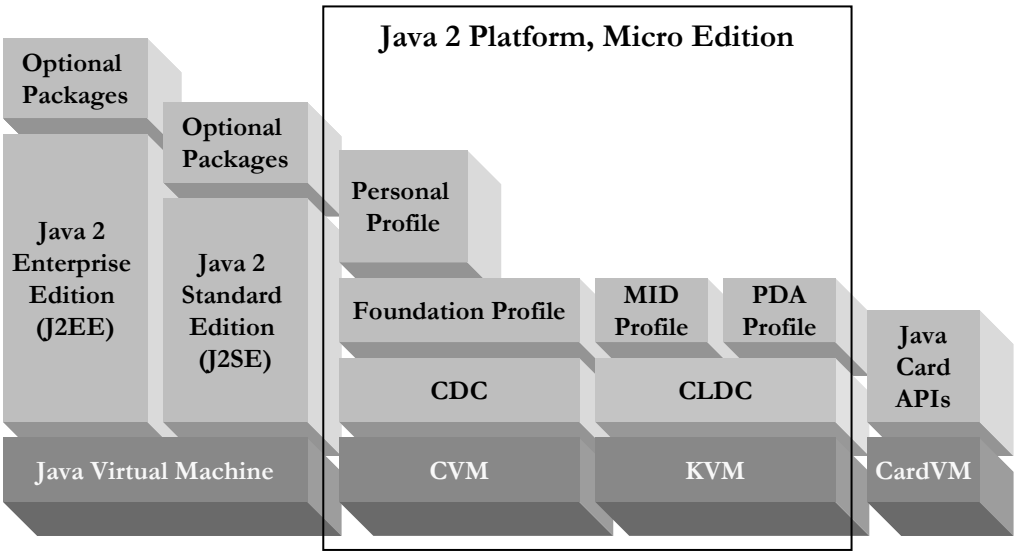


Figure 3-1 Java 2 Platform, Micro Edition

The configuration layer covers the most fundamental features in the java environment and defines the minimum features of the java virtual machine and the java class library available on all devices in a wide spectrum of capabilities, also called a horizontal market. These features are automatically included in the profiles that extend this particular configuration, and are assumed to be present for profile implementation on all devices. Today, two configurations are defined in J2ME, the Connected Device Configuration (CDC) [JSR-36] and the Connected, Limited Device Configuration (CLDC)[JSR-30]. The two different configurations is targeted at two broad categories of products:

- CDC is developed for shared, fixed, connected information devices. This category of devices have a large range of user interface capabilities, memory budget in the range of 2 to 16 megabyte, and persistent, high-bandwidth network connections, most often using TCP/IP. Examples are TV set-top boxes, Internet TVs, Internet-enabled screen phones, high-end communicators, and automobile entertainment/navigation systems.
- CLDC targets personal, mobile, connected information devices like cell phones, pagers and personal organizers. These devices have very simple user interfaces, minimum memory budget starting at about 128 kilobytes, low bandwidth, and intermittent network connections where the network often is not based on the TCP/IP protocol suite.

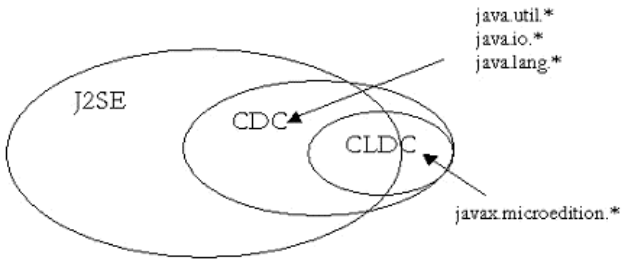


Figure 3-2 Relationship between J2ME configurations and Java 2 Standard Edition

One of the keys to get Java running on small devices is to reduce the size of the runtime classes installed with the runtime environment. J2ME was designed from scratch, with focus on compatibility with the

other Java platforms. Compared to these, J2ME does only include the most necessary classes and represents a subset of the J2SE runtime classes. By definition, all J2ME configurations must adhere to a nested relationship. In other words, the CLDC fits completely inside the CDC. There are no classes, methods or other functionality in the CLDC that are not also in the CDC. This relationship, and the relationship to Java 2 Standard Edition, is shown in Figure 3-2.

The Profile Layer defines the minimum set of Application Programming Interfaces (API) available on a particular family of devices, i.e. a specific vertical market like mobile phones and PDAs. It is typical that the profile includes class libraries that are far more domain-specific than the class libraries provided in a configuration. Applications are written for a particular profile, and uses implicit the configuration since a profile is implemented on top of a particular configuration. By designing this layered design, the application is portable to any device that supports the specific profile it is written for. A device can support multiple profiles.

Figure 3-3 The J2ME tree

Several profiles are under development, and version 1.0 is already released for some. The relation between the different configurations and profiles are shown in Figure 3-3, and explained below.

- For CDC three profiles are being standardized at the moment, the Personal Basis Profile [JSR-129] and Personal Profile [JSR-62], the RMI profile [JSR-66] and the Game Profile [JSR-134]. All these are based on the Foundation Profile [JSR-46], as shown in the figure above.
- CLDC have two main profiles, and several specialized profiles that add functionality to the main profiles. The two main profiles are the Mobile Information Device Profile (MIDP) ([JSR-37], [MIDP1.0]), and the Personal Digital Assistant Profile (PDAP) [JSR-75].

The focus for this assignment is the CLDC and MIDP, and so the rest of this chapter will provide information about these two specifications. Many of the technologies discussed in later chapters will also be available for CDC and related profiles.

3.1.1 Connected, Limited Device Configuration (CLDC)

The intention of Connected Limited Device Configuration (CLDC) is to serve as the lowest common denominator building block for various kinds of resource-constrained, java powered devices. CLDC needs to be complemented by profiles since it is not a complete, self-sufficient solution. For instance, all user interface aspects are outside the scope of the CLDC specification.

The primary topics addressed by the CLDC 1.0 specification are the following areas:

- **Java language and virtual machine features**
- **Core Java libraries** (java.lang.*, java.util.*)
- **Input/Output:** Classes for handling streams of different types
- **Networking:** general framework for network connection, presented in section 3.2.
- **Security:** Further discussed in section 3.6.
- **Internationalisation:** Handles different encodings of character data correctly.

Version 1.1 of CLDC will not include significant extensions to the libraries in CLDC 1.0, but has as goal to make CLDC compliant with the Java Language and Virtual Machine Specification. While CLDC 1.0 does not support floating numbers, the expert group intend to include support in the revised version. [CLDC1.1]

3.1.2 Mobile Information Device Profile

Mobile Information Device Profile (MIDP) is a set of Java APIs that, together with the Connected Limited Device Configuration (CLDC), provide a complete J2ME application runtime environment targeted at mobile information devices (MIDs), like mobile phones. The main goal of the MIDP expert group was to establish an open, third party application development environment for MIDs. The memory and network requirements of a MIDP device are:

- 128 kilobytes of non-volatile memory.
- 8 kilobytes of non-volatile memory for persistent data created by the applications.
- 32 kilobytes of volatile memory for the Java runtime, that is to say the Java heap.
- The device should support two-way, wireless networking with limited bandwidth.

To achieve broad portability the MIDP API considers absolute requirements, i.e. no optional requirements, which are:

- **Applications** (i.e. defining the semantics of a MIDP application and how it is controlled)
- **User interface**, or UI (includes display and input)
- **Persistent storage**
- **Networking** (HTTP 1.1)
- **Timers**

MIPD introduces a new application model, which omits the constraints of having one unique launch point. It introduces a state machine where the application can be in one of the three states active, paused and destroyed, and transit between the states. The central goal of this application model is to provide support for controlled sharing of data and resources between multiple, possibly simultaneously running MIDlets, which is the basic unit of execution in MIDP.

MIDlets can be packaged as Java Archive (JAR) file, either as a single application or as a suite of MIDlets, to be executed on a java enabled device. An optioinal file called the Java Application Descriptor (JAD), which is used to manage the application, can accompany the JAR file. Retrieval of the application can be done either by Over-The-Air (OTA) downloading, or by ordinary downloading to the pc and then usage of a cradle or IR-connection, if available. OTA is not formally included in version 1.0, but will be part of version 2.0 of MIDP [MIDP2.0].

MIDP 2.0, which is soon to be released, also includes other improvements. This includes security in domain and network, additional networking functionality, extensions to the user interface and inclusion of a small sound API. It was also suggested to include a small XML parser, but this is left out for further investigation.

User interface extensions includes extension to the low-level user interface to allow greater game functionality, and layout control for larger screen sizes. A basic sound API called Mobile Media API [JSR-135] is also included, and allows easy and simple access and control of basic audio and multimedia resources. The security issues added will be discussed in section 3.6, while the extension of the network abilities will be covered in chapter 3.2.

3.2 Networking in J2ME

Wireless Java applications are, by their nature, network-centric. However, the devices that these applications run on are less predictable. The precise nature of the network connection depends both on the device and on the services provided by the network to which it is connected. Some may be directly connected to the network, while others are only able to access it through a gateway. Regardless of this, wireless Java devices that conform to the MIDP specification is required to provide the illusion that it is directly connected to the network.

Networking capabilities in J2ME is provided by the Generic Connection Framework defined in CLDC. The framework gives a consistent way to access and organize data in a resource-constrained environment. The approach used has the benefit that the application code stays the same regardless of the kind of connection that is used. Implementation of the protocol definition is left to the different profiles, for example supports the first version of MIDP the HTTP-protocol.

3.2.1 Overview

Requirements for the networking and storage libraries vary from one resource-constrained device to another, and different networks demand for different types of communication. The requirements for having a small footprint J2ME system has therefore led to the generalization of the J2SE network and I/O classes, that is extensible, flexible and coherent in supporting new devices and protocols. Instead of using a collection of totally different kinds of abstractions for different forms of communications, a set of related abstractions are used at the application programming level.

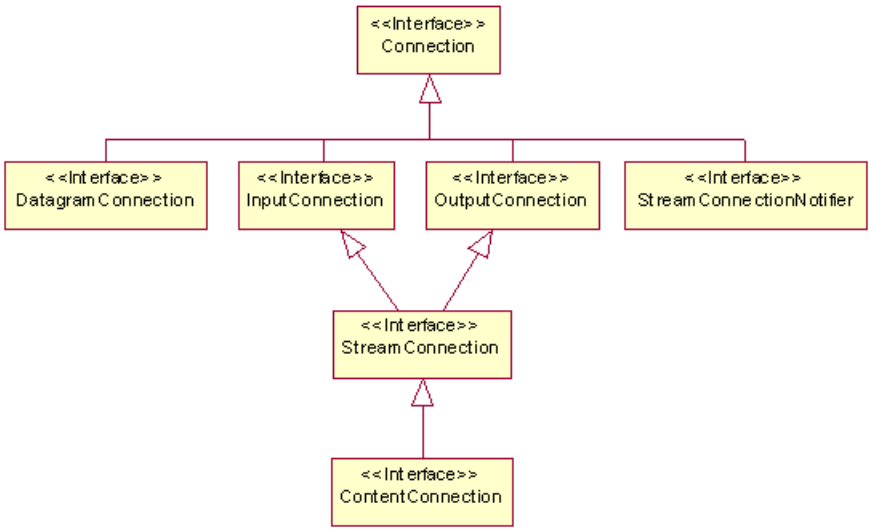


Figure 3-4 CLDC Generic Connection Framework

This platform-independent framework provides its functionality without dependence on specific features of a device. It provides a hierarchy of connectivity interfaces as shown in Figure 3-4, but it does not implement any of them.

All connections are created using a single static method in a system class called Connector, and if successful, the method returns an object that implements one of the generic connection interfaces. This object is created on background of the Uniform Resource Indicator (URI) provided when calling the static method. The URI syntax is defines in IETF standard RFC2396 [RFC2396]. It is composed of three parts: a scheme, an address, and a parameter list. The general form is:

<scheme>:<address>;<parameters>

The scheme identifies how the connection is made, for example socket, http, file or datagram, while the address identifies what to connect to and the parameters identify other information that is required by the protocol to establish a connection such as a connection speed. Examples of URI are http://www.anyaddress.com:8080, socket://localhost:8080, and datagram://127.0.0.1:8099 [White].

Topley discusses some of the pitfalls of MIDP in his article [Topley], and point out that there are differences between HTTP clients written for J2SE and MIDP HTTP clients. A porting is not possible, since the MIDP HTTP support currently presents a lower-level interface than its J2SE counterpart. This results in that MIDP client will have to handle details such as a server redirection, which is taken care of automatically for desktop devices. One must also always be aware of the resource limitations imposed by MIDP devices and restructure the code accordingly.

3.2.2 Protocol implementations for MIDP 1.0

The network support included in MIDP is based on the Generic Connection framework from CLDC. A requirement for MIDP 1.0 is the support for accessing HTTP 1.1 servers and services. The reason behind the HTTP support is the fact that MIDP-enabled devices may not have built-in support for the IP protocol, and HTTP can either be implemented using IP protocols (such as TCP/IP) or non-IP protocols (such as WAP).

Because of the variety of wireless networks, a gateway might be required that can bridge between the wireless transports specific to the network and the wired Internet. This gateway will be responsible for URL naming resolution so that the device may access the Internet. This is shown in Figure 3-5. The application is not required to know about what sort of network it is using, even though it might take advantages of such knowledge to optimise transmission.

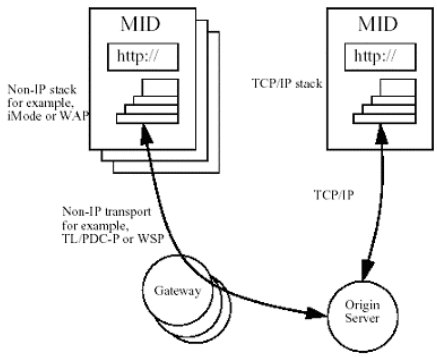


Figure 3-5 HTTP Network Connection [JCP-37]

The interface HttpConnection provides the additional functionality to request headers, parse response headers, and perform other HTTP specific functions. Each device implementing the MIDP must support opening connections using the http URL scheme defined by RFC2616 [RFC2616]. This includes the full specification of RFC2616 HEAD, GET and POST requests, and the absolute forms of URLs.

Using HTTP makes the device able to call Internet services that are based on the HTTP programming model, such as CGI script, PHP and Servlets as shown in Figure 3-6.

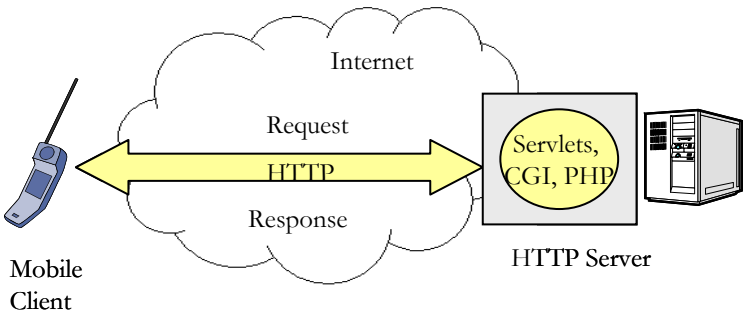


Figure 3-6 Mobile Client using HTTP to connect to Internet Services

It is up to those implementing the MIDP to support other protocols. The only requirement is that the protocol should use the Generic Connection framework. An example is Motorola's Accompli 008 that provides support for Sockets [Acc008].

3.2.3 Protocol support in the future

The next version of MIDP will introduce several new types of connection for mobile devices, but not all are mandatory yet. Some will remain optional since the type of connection is dependent on the resources of the MIPD-enabled devices. And as long as these devices have today's constrains, HTTP is the only protocol suitable.

Datagram and sockets are introduced in MIDP 2.0, and will be discussed next. In addition some actions are taken towards more network security, and MIDP 2.0 [MIDP2.0] will provide a framework to provide both secure http connections (HTTPS) and secure socket connections. This is covered in section 3.6.

Sockets

Sockets provide the functionality to establish connection between two systems, and communicate as if the connection was a stream. This is a primitive, but lightweight and useful method of communication. As sockets has low overhead, this can be one of the fastest methods of exchanging data and issuing commands between two systems [White].

The drawback is that sockets only define the connection and the low-level data transport mechanism, and leaves the client and sockets listeners to define a protocol of how the two systems communicate. In other words, the sockets provide connection, but the format of the exchange information is left to the implementer. The result is that there are few restrictions on what you can do with sockets, but everything you do will need to be determined and built.

Sockets are useful in cases where speed is more important than adhering to open protocol standard since using sockets probably means you will be implementing a proprietary data transport mechanism. But some protocols, like HTTP, are often implemented using sockets. If open standards is an issue in the development, and the developer is not in control of both the server and the client, sockets may not be a good way to implement communication capabilities in the application. Exceptions exist, though. As sockets purely provide the transport mechanism, another data format or protocol could be used in conjunction with sockets, for example XML. Using XML allows the application to take advantage of sockets, while using non-proprietary or publicly defined XML schema. This still requires coordination between client and server application to ensure that they both support the same connection types.

Version 2.0 of MIDP introduces both a socket class and a server socket class, making the mobile java-enabled device capable of exchanging information without use of a middleman or relay. Sockets can both be two cellular phones that send messages to each other, or a mobile device that connects to a J2SE or J2EE server application.

MIDP applications using sockets will still be dependent on the infrastructure to find the server and vice versa. If the mobile device cannot connect to the wireless network, the application would not be able to exchange messages between client and server.

Datagram

Datagram are designed for sending packets of data over a network. The client is not required to set up a connectin to the server before sending packages, i.e. communication is connectionless meaning that there is no established channel between the sender and receiver. All datagram transmissions are considered successful, and so datagram allows data to be sent regardless of whether the server the message is intended to even exists. Thus the connection is consideres unreliable, since a packet that gets lost is not resent automatically by the protocol implementation. The result is that there is no guarantee that the packets will arrive in the same order they were sent if at all, and the protocol does not provide support for reassembling data packets in right order. This in contrast to sockets where the communication is connection oriented, meaning that the connection is set up at initialization and all packets are reassembled and retransmitted if necessary. If a server does not support sockets or is not listening for socket connections, an exception will be thrown. As a result, data sent using sockets is considered reliable. Features of handling the unreliable nature of datagram might be implemented by the application itself.

One primary advantage of using datagrams is the speed. Datagram has no overhead of ensuring that packets arrive in the correct order or that they arrive at all. For some applications the speed is more important than the data integrity, such as audio streaming or video streaming where a missing data packet may appear as static. Although static is not a desirable feature in such applications, the alternative would require the application to wait for all the data to arrive and to place it into the correct order based on how the packets were sent before the data could be officially received. This speed degradation is likely to be unacceptable in applications that are streaming audio or video content.

Several datagram protocols are available, but the most common is the User Datagram Protocol (UDP). There exist interfaces that are designed to allow implementations of different types of datagram protocols, like IP and WDP along with proprietary beaming protocols that take advantage of the packet nature of datagram for transmitting data. UDP requires simpler headers and metadata than TCP as a result of the unreliable nature, and is therefore most useful when speed of delivery is crucial. In the J2ME environment, datagram can be useful due to their simplicity as a lighter weight data transport alternative to TCP. For example might it be useful when beaming data over an Ir port between two devices.

Another feature of datagram is that the programmer controls the packet size of the transmission. If you want to send a large amount of data in a single packet, you can (up to 64kB). You can also send a single byte in a packet.

3.3 Service scenarios for MIDP-applications

After being through different protocols for data transfer, it is time to focus on the different possibilities in which services, or applications, can be supported in the J2ME environment. MIDP 2.0 is not yet released, and not supported in any device, and so this section focuses on the network possibilities related to MIDP version 1.0.

Several possible scenarios exist, and the four main scenarios are presented in this section. All uses the network in some way, even though not always in the execution of the service. They differ in advantages

and limitations, and some might not be able to be realized directly yet because of the limited network support of MIDP 1.0

3.3.1 Standalone service downloaded to the mobile device

The application is available from remote servers, and the mobile terminal establishes a connection and downloads the application he desires, as shown in Figure 3-7.

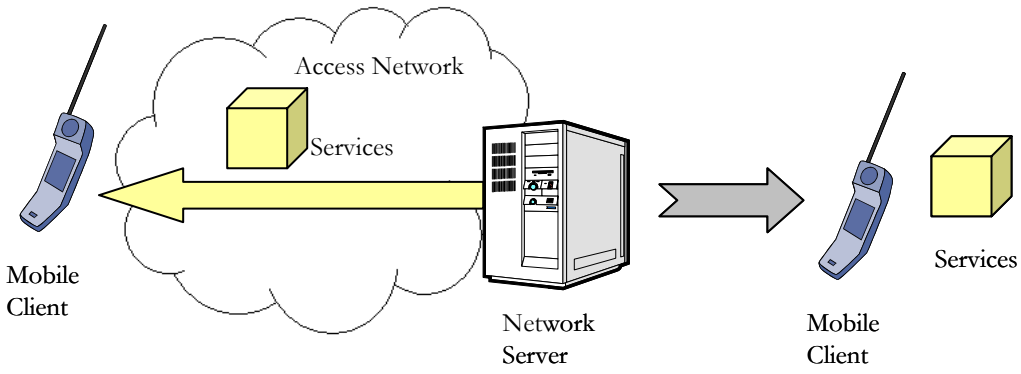


Figure 3-7 Standalone service downloaded to the mobile device

The application is installed and configured on the mobile device, and is executed on the mobile device without network connection. A game is an example of such an application. Because of the lack of network connection dependence, the service can execute independent of whether the network is available or not. The drawback is the lack of network connection, which suits some applications like games. But games are also likely to gain advantages of a network connection to play against other players in a multi-player game.

3.3.2 Service execute on a remote server

Services are provided and executed on remote server after the client has established a connection. After execution the result is transferred to the client, and presented to the user. The remote server may either be in the Network Operators domain or on the Internet. Browsing a website is an example of this kind of service. This scenario is visualized in Figure 3-8.

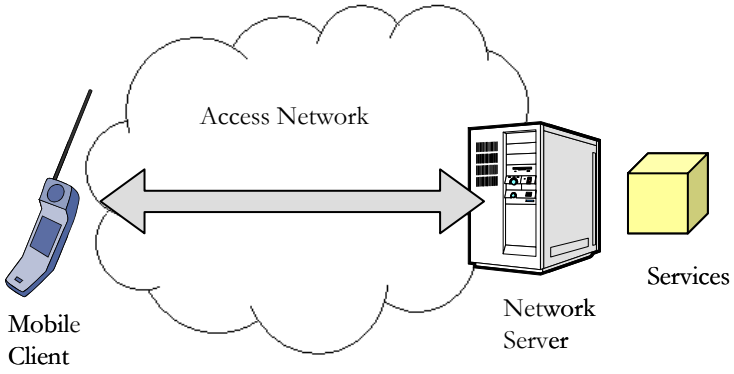


Figure 3-8 Service execution on a remote server

Because of the limited resource of a mobile device, this is the solution for heavy computational tasks

since all the computation is left to the more powerful server, and only the result is returned to the mobile device to deal with. As a result, almost no resources are used on the mobile device. The disadvantage is that the mobile device is dependent upon a server to execute a task, and if the device does not have connection to the network it will not be able to execute the service.

3.3.3 Service client downloaded to the mobile device

The user downloads an application that acts as a local client. This client-application interacts with the service via a connection to a remote server where the service is provided and executed, as visualized in Figure 3-9. This download mechanism is provided by Sun's OTA [Lyng2001]. An email client is an example of a service.

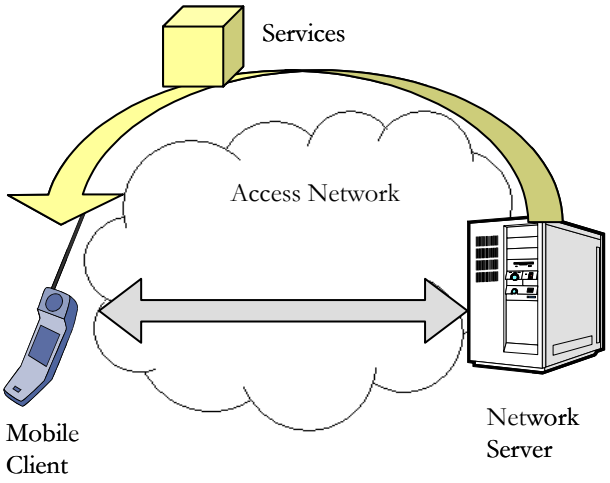


Figure 3-9 Service client downloaded to the mobile device

The advantage of this scenario is that the heavy computational task are left up to the more powerful services, but the client can itself do minor computation, and is so less dependent upon the server to complete a task. It also creates possibilities for the client to be part of the enterprise, and makes the user able to store information at one place and fetch it independent on device, time, and place.

As the previous scenario, this service scenario creates a dependence on a central server, and as a result, the service is dependent on network connection.

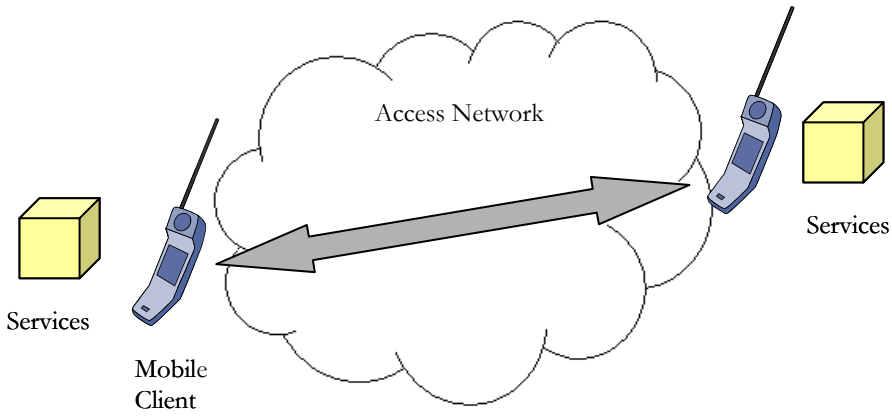


Figure 3-10 Mobile device to mobile device services

3.3.4 Mobile device to mobile terminal services

Mobile devices might wish to establish connections with each other to provide, receive and use interactive services. The services may be downloaded to the devices via the interaction of the mobile devices, or a combination of one of the above-mentioned scenarios. The services may execute locally without relying on servers to support the service. Examples of such services are interactive games and sharing of calendar information.

As of today this sort of communication has to go through one or more servers in the network that acts as relays on behalf of the mobile device. This sort of architecture, middleman architecture, is described in the next section. The hope for the future is to enable direct connection from one mobile device to all other mobile device, independent of infrastructure like today's GSM network. This scenario will be covered in detail in later chapters.

3.4 Middleman Architecture

To overcome the network capability limitations of CLDC and MIDP, a so-called middleman architecture or relay architecture might be used. This architecture is shown in Figure 3-11 and discussed in [Qusay], and is used to enable the mobile device applications to use other network technologies than HTTP.

CLDC and MIDP have no support for socket or datagram connection. In addition, the K Virtual Machine (KVM) does not support all the features of the Java language and the virtual machine, both because they are too expensive to implement and their presence would impose security issues. As a result, there is no support for object serialization, and consequently there is no support for Remote Method Invocation (RMI) either.

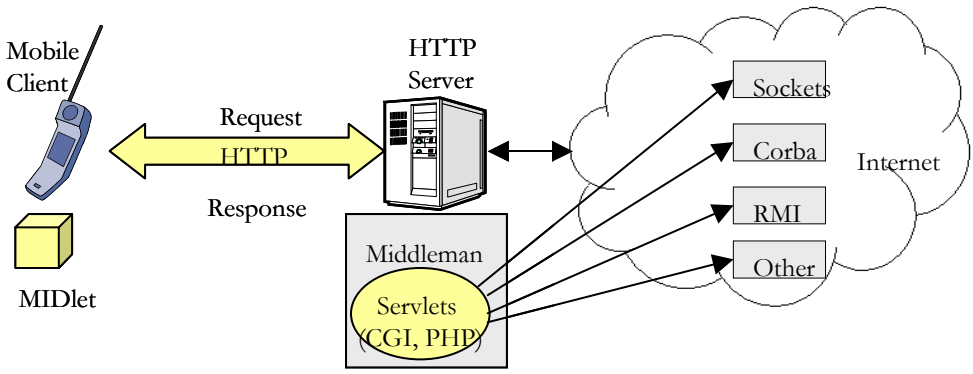


Figure 3-11 Middleman Architecture

Figure 3-11 shows how a MIDlet connects to a Servlet using HTTP. The Servlet can then connect to services using sockets, CORBA, RMI or other java-technologies. To access other services, both CGI scripts and PHP might be used. This way, a MIDlet will be able to participate in distributed environment and become part of a network, and makes the use broader than if the terminal should access services directly via HTTP.

Advantages of the middleman architecture are

- Not all devices are IP-enabled
- It promotes loose coupling and therefore simplifies interactions between objects.
- Enables the user to use RMI, CORBA, or any distributed technology from a MIDP-enabled device.

The drawback is that the MIDlet must know the address to a relay, or the user must supply the application with the address. Both solutions constitute an inflexible implementation.

3.5 New network technologies for J2ME enabled devices

Several network technology is under development and will be provided on embedded, mobile, J2ME enabled devices in the future. Some is under development in the Java Community Process (JCP) [JCP], while independent communities develop others.

Packages from JCP include the usage of Short Message Service (SMS) and supplementary technologies. Others includes a Location API that might be useful, and a SIP API, which all are of minor importance and is therefore covered in Appendix A. Several additional packages and profiles are under development for J2ME, and several are about to be finished. An overview of the different Java Specification Requests (JSRs) that relates to J2ME can be found at [J2ME_spec]. An API for Bluetooth was released in Spring 2002, and last part of this subchapter will give a short introduction to the main features.

Jini and JXTA network architectures are each developed by a community, the Jini Community and the Jxta community, respectively. Both introduces something new to networking, and are part of the new paradigm called peer-to-peer network technologies, and both were originally developed for devices running J2SE or J2EE, but are lately supplemented with compressed versions to be used with J2ME. While Bluetooth covers peer-to-peer in all layers of the protocol stack (including ad-hoc), JXTA and Jini aim at the application layer.

The hope for next generation of mobile phones is independence of transmission technology, i.e. that phones themselves finds the most appropriate bearer technology for his position at the time, and when moving out of range, or the signal from another bearer technology is stronger, switch seamlessly over without user interaction or the user noticing anything.

3.5.1 Jini

Jini differs from all known network technologies. It defines all devices on the Internet as services, and the technology is developed around this concept. A device should be able to be part of a Jini network, and thereby be able to request services from the network. One example might be a printer, which is part of the Jini network. A mobile device requires printing out a mail, but it has no desire to install drivers, it only wants to use a printer near itself. It connects to the Jini network and asks for a printer, and is given the reference to the printer. It is then able to ask the printer to print out the mail for it, without any configuration issues. A more in-depth presentation and discussion of Jini, both in general and for J2ME devices, will be provided in chapter 4.

3.5.2 JXTA

The JXTA project defines a set of protocols that makes it possible to operate on a peer-to-peer basis and a message-format to be used to communicate. As a result, JXTA is independent of implementation language and platform. The reference implementation is in Java, but there are projects going on to implement the protocols in both C and other languages. A more in-depth presentation and discussion of JXTA, both in general and for J2ME devices, will be provided in chapter 5.

3.5.3 Bluetooth

Bluetooth is a wireless point-to-point networking specification, intended to replace the cable(s) connecting portable and/or fixed electronic devices. The key features are robustness, low complexity, low power, low cost, and high-speed. It is intended for use as a personal-area network (PAN), connecting devices such as cell-phones, PDAs, printers, and headsets together when they are near each other.

The Bluetooth standard is divided in two groups: core and profile. The core specifications describe the details of the various layers of the Bluetooth protocol architecture, from the radio interface to link control. Related topics are also covered, such as interoperability with related technologies, testing requirements, and a definition of various Bluetooth timers and their associated values.

The second group, the profile specification, is concerned with the use of Bluetooth technology to support various applications. Each profile specification discuss the use of the technology defined in the core specifications to implement a particular usage model, and include a description of which aspects of the core specifications that are mandatory, optional and not applicable. The purpose of a profile specification is to define a standard interoperability, so that products from different vendors that claim to support a given usage model will work together. In general, profile specifications fall into one of two categories, cable replacement or wireless radio.

The Java APIs for Bluetooth Wireless Technology [JSR-82] is developed for usage on CLDC and covers pure transmission of data. The intention of this API is to support registration of services and discovery of devices and services. It is capable of establishing RFCOMM, L2CAP and OBEX connections, and conduct all these activities in a secure fashion.

The protocols covered are:

- **L2CAP** (Logical Link Control and Adaptation Protocol): Resides in the data link layer, and supports higher-level protocol multiplexing, packet segmentation and reassembly, and the conveying of quality of service information.
- **RFCOMM**: A protocol that provides multiplexing and transport protocol, and is based on the ETSI standard 07.10. It is the cable replacement protocol included in Bluetooth, meaning that it has support for serial cable emulation and other cable emulations.
- **SDP** (Service Discovery Protocol): Defines a protocol for locate services provided by or available through Bluetooth devices, and to determine the characteristics of those available services.
- **OBEX** (Object EXchange): The purpose of the OBEX protocol is to enable exchange of data objects. Typical example could be an object push of business cards to someone else.

The profiles included in the J2ME implementation of Bluetooth are listed below.

- **Generic Access Profile (GAP)**: Defines the generic procedures related to discovery of Bluetooth devices and link management aspects of connecting to Bluetooth devices. It also defines procedures related to use of different security levels. In addition, this profile includes common format requirements for parameters accessible on the user interface level.
- **Service Discovery Application Profile (SDAP)**: Defines the protocols and procedures to discover services registered in other Bluetooth devices and retrieve any desired available information relevant
- **Serial Port Profile (SPP)**: Defines the requirements for Bluetooth devices necessary for setting up emulated serial cable connections using RFCOMM between two peer devices.
- **Generic Object Exchange Profile (GOEP)**: Defines the requirements for Bluetooth devices necessary for the support of the object exchange usage model.

3.6 Security

The security in MIDP is limited, and as a result MIDlets is a potential threat to users. Some improvements will come with the next version of MIDP, and these will further limit the risk. But other risks still exists, not so much on the mobile device itself, but in communication with entities on a network.

Even though MIDlets have great opportunities when it comes to networking, a lot of applications will not be possible without some form of data security. Any information transmitted over wireless links is subject to interception. Some of that information could be sensitive, such as credit card numbers and

other personal information. The solution needed depends on the level of sensitivity. To provide a complete end-to-end security solution, you must implement it on both ends, the client and the server, and assure yourself that intermediary systems are secure as well. One solution to consider when handling highly sensitive information is encryption: the sender encrypts the data before transmitting it over the wireless link; the authorized receiver receives the encrypted data and decrypts it using a key provided.

This section first considers the limitations of the current version of MIDP, and which improvements that are investigated in MIDP 2.0. Then the attention is turned to network communication and cryptography, and what security packages that are available to MIDP, or soon to come.

3.6.1 Application security on the device

The security in Java 2 Standard Edition (J2SE), though both powerful and flexible, exceeds the amount of memory size available for virtual machines supporting CLDC. It also requires a certain amount of administration that may be beyond the knowledge expected of a mobile device user.

CLDC and MIDP introduce low-level KVM security and application-level security. The low-level security is concerned with verification of java classes, while the application level security focuses on the execution of application on the java-enabled device.

To ensure that the virtual machine (KVM) does not do any harm to the device in which it is running, the java classes must be verified. This verifier ensures that the Java class file cannot contain references to invalid memory locations. In J2SE this verification is done prior to execution, but sine MIDP enabled devices have memory constraints, the classes must be pre-verified, meaning that the verification process takes place before downloading the application. Prior to execution, the in-device verification process is carried out, using the information generated by the pre verify tool.

When CLDC was designed the security model was kept simple and it allows for more comprehensive security solutions in later versions of the CLDC specification [Lyng2001]. The KVM includes a simple "sandbox" where the java application must run, and ensures a closed environment in which the application can only access those APIs that have been defined by the configuration, profiles, and licensed open classes supported by the device. But a security checking of API calls, which is available in J2SE, is not included, with few exceptions. As MIDP 1.0 does not support any additional features than those already provided in CLDC, there practically does not exist any security for the user to deal with [KTopley]. The result is that MIDlets is a potential threat to the user.

The limited security included ensures that MIDlets have no access to information stored on the phone, such as telephone number lists or calendars, and a MIDlet is not allowed to take directly control over the device. The data stored by the MIDlets is private to that MIDlet and its suite, so it can only harm its own data.

3.6.2 Network security and cryptographic solutions

When transmitting data over a network, it is possible someone is eavesdropping at various points in the network, i.e. listen in on the data that is transmitted. This is the reason that something is needed to keep sensitive data from being stolen. One solution to this problem is use of cryptography.

Cryptography is the science of secret writing, and is a branch of mathematics that is based on the idea that certain kinds of mathematical problems are hard to solve. As research in mathematics continues, it is very possible that someone will discover a way to solve most of the modern cryptographic algorithms. But for today, it provides protection for sensitive data, and there are not many acceptable alternatives. ([Knudsen], [Gollmann]).

Eavesdropping on insecure communications links can be counteracted by services and mechanisms from communications security. Important security services, or aspects include:

- **Confidentiality**: Other people should not be able to see sensitive information that is sent over the network
- **Integrity**: Ensure that the data is not getting changed or corrupted in any way.
- **Authentication**: The process of proving identity.

Cryptography provides solutions for each of these security aspects; Integrity Check Functions, also called message digests, digital signatures and encryption algorithms, also called chipers ([Knudsen], [Gollmann]).

- **Integrity Check Functions**: Applies a hash function to a large piece of data, to transform it into a small piece of data. If only one bit of the original data is changed, it will result in a totally different value. The result of applying a hash function has several names, like hash value, message digest and checksum. It is also sometimes referred to as a digital fingerprint. An example of a hash function in use is the Secure Hash Algorithm (SHA-1).
- **Digital signatures**: A digital signature is like an integritycheck function except it is produced by a particular person, the signer. The scheme consists of a signature algorithm and a verification algorithm. To create the signature, the signer must have a private key, while a corresponding public key can be used by anyone to verify that the signature came from the signer. Certificates are really just an extension of digital signatures, which is a document signed by some authority, that proves your identity. It's like a driver's license, except it is based on digital signatures. Examples of digital signatures scheme are El Gamal Signatures, Digital Signature Algorithm (DSA) and the RSA algorithm.
- **Encryption algorithms (Chipers)**: An encryption algorithm can be used to encrypt or decrypt data. Encryption means to make an unreadable mess of the supplied data, called cipher text. The cipher text can be decrypted back to the plaintext provided in the first place. This requires use of keys. Using keys, the cipher text will be different for each key. In symmetric algorithms, the same key is used for encryption and decryption, while in asymmetric algorithms, also called public key algorithms, different keys are used for encryption and decryption. Chipers can operate in different modes that determine how plaintext is encrypted into cipher text. This affects the use and security of the cipher. Examples of chipers are Data Encryption Standard (DES), which is symmetric, and RSA encryption that is asymmetric.

Installation of MIDlets should only take place when the software comes from trusted sources, but today there are no mechanisms for the user to be completely sure who is actually providing a MIDlet or that the MIDlet code has not been interfered with on the route to the device. J2SE includes an authentication mechanism, i.e. public key cryptography and certificates, but this is not included in the MIDP 1.0 specification. MIDP 2.0 introduces an improved domain security model that has been supplied to include signing of applications and verification of certificates. In this way the user should be able to verify the MIDlet provider.

Cryptography support is provided in J2SE through the Java Cryptography Architecture (JCA) and the Java Cryptography Extension (JCE). But both are too heavy for the MIDP platform. So far, there does not exist any solutions to cryptography in J2ME from Sun. Solutions still exists, like the Bouncy Castle cryptography package [Bouncy], an open source effort based in Australia. Some claim it is a wonderful piece of work, featuring a clean API and a formidable toolbox of cryptographic algorithms [Knudsen]. Bouncy Castle offers a lightweight J2ME distribution of their software, which other open source cryptography packages lack.

There exist security solutions to some of the problem encountered in J2ME MIDP that does not involve handling the security on application level. It is still cryptography, but it is handled as part of the protocol stack responsible for sending the data over the network before the data is sent. E-commerce applications uses the Secure Socket Layer (SSL) developed by Netscape, and the hope is that it will work nicely for

m-commerce as well [Mahmoud]. Sun has been working on a stripped-down version of SSL called kSSL, and has already delivered it as part of the MIDP 1.0.3 reference implementation and the J2ME Wireless Toolkit 1.0.3. kSSL will be included in the MIDP 2.0 specification, and this will then be able to support the secure version of the HTTP protocol, HTTPS, and secure socket connections ([MahLor], [Gupta]).

To further improve the security in J2ME, a security API is under development at Java Community Process (JCP) [JCP], JSR 177 Security and Trust Services API for J2ME [JSR-177]. It is proposed as an optional package to be used together with several J2ME profiles, and anticipated schedule for the specification is first or second quarter of 2003. The objective is to define a collection of APIs that provide security services to J2ME enabled devices. This will make the device become trusted, and as a result provide security mechanisms to support a wide variety of applications based services, such as access to corporate network, mobile commerce, and digital rights management. Many of the services rely on the interaction with a security element in the device for secure storage and execution. This might be secure storage to protect sensitive data, such as the user's private keys, public key certificates, service credentials and personal information. Other areas might be to ensure a secure execution, such as cryptographic operations to support payment protocols, data integrity, and data confidentiality. And last, add security features that J2ME applications can rely on to handle value-added services, such as user identification and authentication, banking, payment, ticketing, loyalty applications, and digital media play. It is suggested to implement the security element as smart cards, as they are widely deployed in wireless phones, such as SIM cards in GSM phones. As a result the specification will provide an access model that enables applications running on J2ME enabled devices to communicate with a smart card inserted in the device [JSR-177].

3.7 Summary

J2ME is a collection of APIs focusing on consumer and embedded devices. The main components are configuration and profiles, where the configuration covers the most fundamental features in the java environment and the profiles defines additional functionality like user interface, storage and network capabilities. One of the two configurations is the Connected Limited Device Configuration (CLDC), which is the fundament for the Mobile Information Device Profile (MIDP), the profile in focus in the rest of the chapter.

The network capabilities of MIDP is built upon the Generic Connection Framework defined in CLDC that provides a consistent way to access and organize data in a resource-constrained environment. MIDP implements the HTTP 1.1 protocol, but further support of sockets and datagram is to be provided in the next version of MIDP. Services, or applications, can be used and accessed in several ways using MIDP. Both by downloading the application to the device to run locally, to execute all on a remote server and only retrieve the processed data to the mobile device, or a combination of these two approaches. The hope is that in the future, two mobile devices will be able to communicate independent on the infrastructure. Today, MIDlets are dependent on a relay that acts on behalf of it to access a network. But doing so enables the MIDlets to use more advanced network technologies like RMI and CORBA. New technologies that use this architecture are the Jini network technology and the JXTA technology. Use of Bluetooth will remove the dependence on a relay, but will bind the device to use Bluetooth for all network connections.

MIDP also have limited security, especially when it comes to wireless networking. Packages to ensure integrity, confidentiality and authentication are needed, but not available. An API is under development, and in the mean time the open source project Bouncy Castle can provide J2ME with a cryptography package. But adding cryptography to an application or system will not necessarily make it more secure. Comprehensive system-level approach to security also has to be done, and cryptography is just one of the tools in your box. MIDP 2.0 will improve the security by introducing certificates and verification of the MIDelts, and SSL to make secure http and socket connections. But until this version of MIDP is final and supported in all devices, there is limited security against malicious MIDlets.

Chapter 4

Jini Network Technology

Any sufficiently advanced technology
is indistinguishable from magic.
Arthur C. Clark

Jini is the Arabic word for magic, and is a distributed, dynamic networking architecture developed by Sun Microsystems [Sun]. The vision of Jini is that you can plug in any Jini-enabled device into a network, and automatically discover and use the variety of other Jini-enabled devices in your vicinity. Any resources available on the network are available to your Jini-enabled device, as if it were directly attached to it, or the device had been explicitly programmed to use it. Adding new devices is easy, and adding or moving services can be done without extensive configuration. The Jini technology-enabled device can announce itself to the network and tell other devices how to talk to it, and immediately become accessible to other devices in the network. The devices can come together to form an improvised community, with no human intervention. A Jini-enabled device can be a digital camera, a new printer, a PDA, a cell phone and other devices. Imagine printing out your email from your phone on the printer nearest to you at that moment just by choosing the print option, or sending a fax to a friend.

This chapter aims at giving the reader an overview of the Jini network technology, its features, advantages and limitations, as well as usage of the technology for embedded devices. First it will give an introduction to Jini, then present the architecture and core components of the architecture. The rest of the chapter will focus on Jini solutions for embedded, mobile devices like mobile phones and PDAs.

4.1 Introduction to Jini Network Technology

The objectives of Jini are to provide an infrastructure to connect anything, at any time, anywhere, and enable "network plug-and-work", i.e. make every service joining the network available for other users without installation and configuration [Jini]. Hardware and software distinction is abstracted by supporting a service-based architecture, instead of a computer or device network. [Kumaran]

Jini distributed architecture provides spontaneous networking of services, where the term services refers to a set of functionality provided by any entity. An entity can be a computer hardware device, such as computer, printer or scanner, an electronic device, for example audio and video equipment, an electric device like microwave oven, coffee maker or other household appliance, a piece of software, or a combination of hardware and software.

The network technology is an open architecture that enables developers to create network-centric services that are highly adaptive to change. Jini technology can be used to build adaptive networks that are scalable, evolvable and flexible as typically required in dynamic computing environments.

Jini network technology has several advantages and limitations, some were found during the study of the technology, others were pointed out in [Jini]. The advantages are:

- Jini provides an environment for creating and managing dynamically networked components, applications, and services that scale from the device to the larger enterprises.
- The technology is based upon mobile code, i.e. it moves data and executables via a Java object between entities and so extends the Java programming model to the network, and ensures a homogeneous environment.
- No user interaction or configuration is required to allow devices on a network to bring services on or off line.
- Enables network self-healing, and services can dynamically join and exit with no impact on the network or the network users, i.e. the community adapts to changes.
- Consumers of Jini services do not need prior knowledge of the service implementation to be able to use the service.
- Jini is a protocol-independent distributed computing architecture. The proxies can interact with its home service using any distributed protocol, for example RMI, CORBA, DCOM or a proprietary protocol.
- Jini offers an open development environment for creating collaboration through the Jini Community, and the technology is available for free.

The limitations of Jini are:

- Assumes that 'Java is everywhere', i.e. all services can be transported as java mobile code and run on a Java virtual machine or have an entity that can run this virtual machine on behalf of the service.
- Assumes that a Jini Lookup Server always is present on the network, witch might not be true. If none exists, Jini will not be able to work.
- Dependent upon TCP/IP, which is most often not supported in wireless, mobile devices.
- The reference implementation from Sun is dependent on the remote method invocation (RMI) technology. This dependence is related to downloading of stubs and enabling of a connection between an interested client and the lookup service.

4.2 Jini Architecture

In the simplest way, Jini can be seen as consisting of three entities; the services, the lookup servers that catalogue available services, and the clients that requires the services. These entities are built up of components and sub components of the system architecture of Jini.

The system architecture of Jini consists of three components that collaborate with each other to achieve the overall system objectives. These components are

- The infrastructure component,
- The programming model component, and
- The service component.

Figure 4-1 shows the relationship between the three components of Jini, and some of the subcomponents. Notice that the infrastructure component and the service component both uses the programming model component, and that the Lookup Service is part of both the Infrastructure component and the Services.

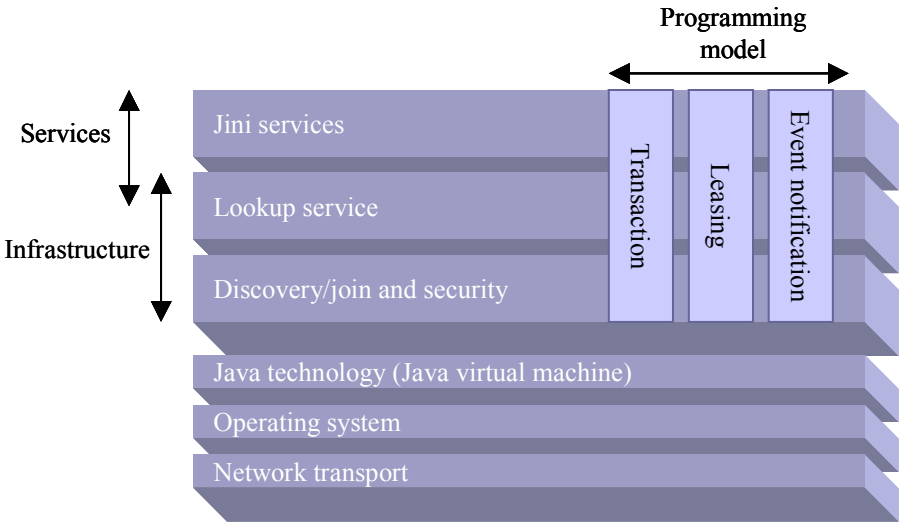


Figure 4-1 Jini component overview

Jini extends the java platform as shown in Figure 4-2. The technology assumes that java is always present on the network, either in the services themselves or as a java-wrapper. In other words Jini requires that each device either run a Java virtual machine (JVM) or associates itself with a device that can execute a JVM on its behalf. This assumption is made since services are identified by Java type, and proxies may need code to download. Service implementations that are non-java can wrap the legacy language in Java and make calls to legacy with Java Native Interface (JNI).

Remote Method Invocation (RMI) is fundamental to the reference implementation of Jini, but the features provided by Jini should be available independent of this package. For example PsiNaptic [PsiNaptic], a communication technology company, has implemented a version of Jini that do not use RMI. This chapter will focus on the reference implementation of Jini, and by doing that also cover the RMI dependence. For other relationships between the component of Jini and Java, refer to Figure 4-2.

	Infrastructure	Programming model	Services
Jini	Discovery Lookup Extended Security	Lease Event Transaction	JavaSpace TX Management Other Services
Java	Java VM RMI Security	Java technology-based APIs Beans Swing	Enterprise JavaBeans Java Naming and Directory Interface JTS

Figure 4-2 How Jini extends the Java platform

The dependence on Java and its features gain many benefits for Jini. For example the Java virtual machine ensures a homogeneous network, and the portable object code gives architecture independence. A dynamic environment is ensured using downloadable code, and impedance mismatch is solved by the unified type system in Java.

4.2.1 The infrastructure component

The core part of the Jini architecture is the infrastructure component. Its goal is to provide mechanisms for devices, services and users to discover, join and detach from the network. The following subcomponents is part of the Infrastructure component [Kumaran]:

- Lookup service component
- Discovery protocol component
- Join protocol component
- Remote method invocation (RMI) environment component
- Security component

Jini lookup service is a repository of available services, where each service is stored as a Java object and made available for clients to download on demand. The lookup service is placed in the lookup server, which client entities assume always is present on the network. The interfaces/operations available on the lookup service is registration, access, search and removal of services.

Jini discovery and join protocols are used to find and join a group of services, and are based on UDP multicast. The three protocols are [Computing]:

- **The multicast request protocol**: Locate one or more lookup servers
- **The unicast discovery protocol**: Used by clients and servers to communicate with a specific lookup server
- **The multicast announcement protocol**: Used by a lookup server to announce its presence on the network. When a lookup server invokes this protocol, clients and services that have registered interest in receiving announcements of new lookup services are notified.

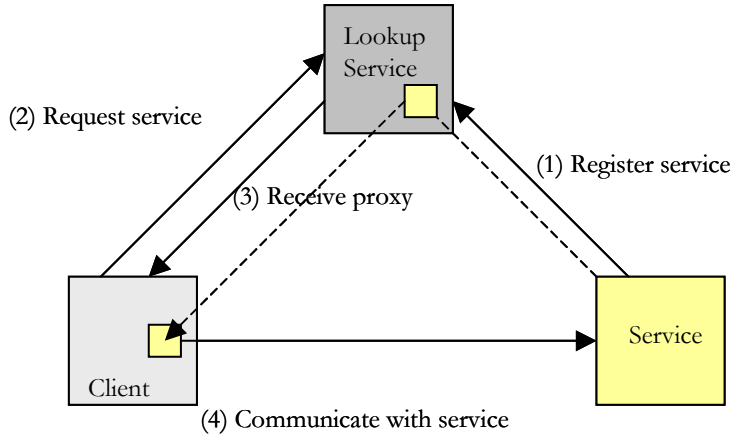


Figure 4-3 Discovery and interaction with a service

Discovery in the Jini network start with the service multicasting a packet with reference to itself, and receives a RMI reference to one or more lookup services. The Jini client locates a service by querying the nearest lookup service, and as response it receives a proxy representing the service. Request to the service is sent to the downloaded proxy, which handles all communication with the remote service. As a result, the usage of a service is independent on transfer protocol, which can change without affecting the client. *Join* is the action of a service placing an object representing its capabilities into the lookup service for other clients and services to find and use.

The discovery, join, and service usage is shown in Figure 4-3. The small dark boxes is the proxy that represents the Service, and the dotted line shown how a service first is places with the lookup service, and

then downloaded to the client. Communication between the proxy and service is shown as a solid-drawn line.

Jini uses Java's *Remote Method Invocation (RMI)* facility for all interactions between either a client or a service and the lookup server (after the initial discovery of the lookup server). The reason to use RMI is that the technology is able to pass entire Java objects and their code and works in any compliant Java virtual machine. RMI also provides automatic serialization, transport, and deserialization of objects, and have robust and configurable security like RMI over SSL and authenticated principal.

Jini security component is based the Java's security model [Kumaran]. This model provides features like digital certificates, encryption, and control over mobile code activities such as opening and accepting socket connections, reading and writing to specific files, and using native methods. Jini services use these security models, while Jini uses the two security aspects:

- File permission: Allows access to all relevant files, such as class files, jar files, and related files.
- Socket permission: Allows access to an open network socket on any port.

The permissions associated with the Java runtime environment is placed in a java policy file, and separate security policy file can be specified to set application-specific policies. In addition to the permission classes defined in Java, Jini also defines one for discovery permissions, which specifies permission to connect to a Jini lookup service that is serving a particular group. Systems administrators can establish different policies depending on where the Java code originated, for example the local file system or a remote machine.

Downloading mobile code gives an opening for viruses, which is also mobile code. So before code can move to another machine in the Jini environment, it must satisfy the client's security policy witch should only allow downloading from trusted machines. Other problems could be the lookup server sending a client to the wrong location since there is no way a lookup server can validate a service's purpose. ([Computing],[ITPro])

4.2.2 The programming model component

Jini programming model is based on the Java application platform to form a distributed extension. This is done by introducing a set of interfaces or components [Kumaran];

- Leasing
- Event notification, and
- Transaction.

Leasing is introduced to solve the problem with partial failure, which in distributed systems can lead to unchecked resource consumption. Traditional solutions have been system administration that has proven error-prone, and costly, and it often happens when it is too late. Distributed leasing is a protocol for managing resources using a renewable, duration-based model for allocating and freeing the resource references. This provides a method of managing resources in an environment where network failures can, and do, occur. Leases are time-based grants of resources or services, according to a contract between grantor and holder that is negotiated for a set period of time. Leases can be shared or exclusive, and can be cancelled, renewed, expire or obtained and manipulated by third parties. In a distributed environment, resources are allocated only as long as continued interest is shown, and the holder of the lease is responsible to renew lease before expiry. The concept of leasing ensures that the network is self-healing.

The distributed event notification component extend Java event model, the JavaBeans component event delegation model, to work in a distributed network. Clients register interests and receive relevant notifications, and as such allow for use of event managers. Various delivery models are supported like push, pull and

filter, and the mechanism uses the distributed leasing protocol. The model allows an event to be handled by third-party objects and recognizes that the delivery of the distributed notification may be delayed.

Jini transaction model is designed to coordinate distributed objects. It is lightweight, supports two-phase commit, and allows the system to handle object-oriented transaction handling. The model does not define the actual mechanisms involved in the transaction, but provides rules for the objects involved in the transaction. In addition it uses the distributed leasing protocol in Jini. An implementation is the Transaction Manager service presented in section 4.2.3.

4.2.3 The service component

An important concept within the Jini architecture is the service components, which denotes the entities that have come together to form the Jini community. Such entities could be either hardware and software, or a combination of hardware and software. A service can be composed of other sub services, and is identified as java objects within the system. Each service has an interface that defines the operations that can be requested by that service, and also reflects the service type. [Kumaran]

The interface representation of services separates the actual implementation of the service from what the service does, and hides that the implementation can change over time and whether it is implemented in hardware or software. It also hides the fact that there might be multiple implementations. The base interface can be extended to combine or add new functionality. For example old clients can use the base interface, while new clients use extended interfaces. This allows for evolution of implementation without changing clients or infrastructure. Only base interfaces need to be common, the community of developers can define new ones.

Services are built over infrastructure components using application components as their building blocks, and can be written by anyone as long as it is in compliance with the Jini specification. Examples of services are:

- The lookup service
- The JavaSpace service, and
- The transaction manager service.

Jini lookup service is part of both the Infrastructure and the Service layer of the Jini Architecture, and was covered in section 4.2.1. The lookup service differs from the other services in that it is the only service that can be found using a discovery protocol (all other services are found through requests to the lookup service). Having the lookup service as any other service is an advantage in that it can be managed like any other service, interaction is the same and it is possible to have multiple instances of the lookup service within the same network without conflicts.

Jini JavaSpace Service [JavaSpace] provides an optional distributed persistence-mechanism for the objects within a Jini community. It is simple, and can be transactional secure. The service uses parallel programming in a distributed system where cooperating software ensembles, and provides high-level communication between Java programs. Benefits are anonymity between applications, uncoupled communication, programs can communicate through time or space, and vast savings in design and development time.

Processes in a JavaSpace program do not interact directly, but indirectly through one or more spaces. A space is a persistent object store that is network accessible, shared, transactional secure and able to store executable content. Elements in a space are called entities, i.e. collections of typed objects. An entity in a space cannot be modified directly, but has to be taken out, modified and then written to the space. Jini is used to locate a space, and in Sun's implementation the spaces are identified by name. An example of a JavaSpace is a distributed array. Each array element is an entry in a space, the entry stores content and position in the array, and protocols are defined to add or remove array elements. As a result, multiple processes can concurrently access the array through the space. JavaSpace have a lease mechanism, which

is based on the Jini lease model, and all operations on the JavaSpace are transactional safe, i.e. all operations are atomic, and based on Jini transaction model. Another feature that is made possibly through use of JavaSpace is ad-hoc upgrading of new software. The user of the software each has contract with the provider for how long the software is available, and can upgrade upon expiring. A software distributor can have multiple versions of the same software running, and place new software in the JavaSpace on upgrading. When users ask for an upgrade, access to the new service might be granted. [JiniJO02]

Jini transaction service is an implementation of the Jini transaction interface, and provides distributed transactions for the distributed objects. The transaction model is a completion protocol that uses two-phase commit, but it does not guarantee ACID[1] properties and leaves it to the individual object to handle them. It also supports nested transaction, leasing, and even notification, and can be used by any application that needs a distributed transaction within the Jini community. [Kumaran]

4.3 The Jini community

The Jini community explores, develops and evolves the Jini network technology. Their website [JiniOrg] share ideas, related source code, documentation and other development work based on Jini technology. The site facilitates collaborative development by providing discussion forums, news groups, chat, mailing lists, and anything else that brings the community together.

Jini community is building the necessary structures in the network technology, while Sun has the overview of the technology and the community. Developers can participate in projects, or create a new project if none of the existing projects covers the developer's interests. All projects are licensed under the Sun Community Source License. New project must be proposed to the community to review the project description and so as to avoid duplication of projects. Each project has an owner and a mailing list, and documents regarding the projects are made available for the community on jini.org [JiniOrg].

Projects of interest in relation to embedded devices are:

- **The Surrogate project**: The goal of the Surrogate project are to investigate the use of third-party computational resources to "Jini enable" devices that can not directly interact with a Jini network, and to define a standard way for doing this.
- **Jini WirelessDevice Project**: The purpose is to create the standard needed in order to make a Jini-enabled wireless device. Focuses on Bluetooth interconnect between the surrogate architecture and a device.
- **The Anhinga Project**: Developing variations of J2ME CLDC, Jini, and JavaSpace technologies that run in ad-hoc networks of small mobile wireless devices such as PDAs, cell phones, pagers, and pocket PCs.

The Jini community creates and agree upon standards, which are intended for use by developers as guidelines when developing software using Jini technology. A standard starts as an initiative from one person or a group of person, and all interested parties may collaborate on the proposed standard. Today, there are three standards, and two of them are the Jini Technology Core Platform Specification and the JavaSpace Service Specification. A Print specification, the Surrogate Architecture specification and the IPInterconnect specifications are waiting to be standardized.

4.4 Jini Surrogate Architecture

Many devices are not capable of participating in the Jini network because of memory, processing and communication constraints, or just because they are not Jini-enabled or have no java virtual machine. Even if a device does support Java technology, for example J2ME, it may not meet all requirements to

1. Atomicity, Consistency, Isolation and Durability (ACID)

support a full Jini implementation. Jini Surrogate Architecture specification [JiniSA] is a Jini community project dealing with Jini enabling of devices. By using the surrogate architecture, devices that could not easily take advantage of Jini technology may now become part of a network of Jini technology-enabled services and/or devices. The surrogate architecture defines a surrogate execution environment and a surrogate programming model, and provides requirements for interconnect specifications.

Figure 4-4 shows the Jini Surrogate Architecture. As Jini, the surrogate architecture relies on mobile object, but the direction of the motion is reversed. Where the mobile objects in Jini are loaded into a device, the objects in the surrogate architecture are moved out of the device and into a surrogate host. These objects, known as surrogate object, are written in the Java programming language, and represent and act on behalf of the device.

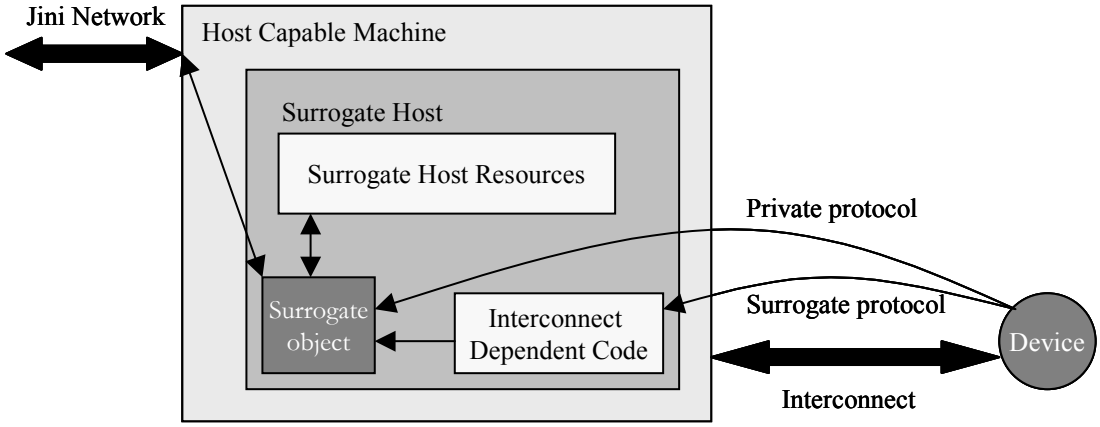


Figure 4-4 Jini Surrogate Architecture

A surrogate host is a framework that provides a Java application environment specially designed for the hosting of surrogates, and so provides a place for the device to participate in the Jini network. The application environment might be composed of J2SE or J2ME CDC + Foundation Profile, Jini APIs, and life cycle support of surrogate objects uploaded into the environment by the limited-capability device. In other words the surrogate host is responsible for managing the life cycle of the surrogate, which consists of the stages device discovery, surrogate retrieval, execution, deactivation, and disposal.

The life cycle starts with the limited-capability devices using a surrogate protocol to discover the nearest surrogate host, or the surrogate discovers the device. It then uploads its surrogate object, also known as a service proxy object, to the surrogate host. The surrogate object, contained as class files in a JAR file, can also be located elsewhere than on the device. After uploading, the device delegates the control thread to the surrogate host, which from then on manages the life cycle of the surrogate object, such as activation, retrieval, execution, deactivation, and disposal. Activation is the act of instantiating the surrogate object, and making the call to begin the surrogate's execution, while execution is the steady state of a running surrogate. When a surrogate is executed it can accomplish anything that other Java programs can do, for example access other Jini services or provide Jini services themselves. During execution, the surrogate object can communicate with devices using any preferred communication protocol, such as TCP/IP, Bluetooth, or any proprietary protocol. The last two stages of the life cycle is deactivation, i.e. stopping the surrogate, and disposal, i.e. remove the objects associated with the deactivated surrogate. ([JiniSAO], [Kumaran])

The surrogate host resides on a host-capable machine, which is a system that allows the downloading of code, can run a surrogate, is part of a Jini network, and is accessible to the entity offering the surrogate. Jini surrogate architecture assumes that there exists a host-capable machine that is connected to both the

interconnect and the Jini network. The surrogate host is responsible for monitoring the interconnect, a logical and physical connection between the surrogate host and the device.

An interconnect strategy is needed to connect the device, and is an addition to the surrogate architecture. The interconnect strategy described the interconnect protocol between the device and the surrogate host as well as interconnect-specific additions to the surrogate programming model. The interconnect protocol defines mechanisms for discovery, retrieval, and availability, and interconnect-specific APIs, which all are device and interconnect dependent. The surrogate host is responsible for implementing the interconnect protocol. Different Interconnect projects are being developed, and some of them are listed below. [JiniSASpec]

- **IPsurrogate**: TCP/IP interconnect ([IPInt][IPIntSpec])
- **WirelessDevice** : Bluetooth interconnect [JiniWD]
- **SmartCard** : Smart Card interconnect
- **IEEE1394** : Firewire/iLink bus interconnect

The surrogate programming model support the life cycle and other interconnection protocol requirements.

The advantage of the surrogate architecture is that it does not depend on any particular capability of a device managed by a surrogate, i.e. it is device independent. They to not need to be java-enabled, and memory, processing and communication constraints will not prevent it from participating in the surrogate architecture. Other advantages is that Jini's "Plug-and-work" goal is fully supported in that discovery, code downloading and leasing of distributed resources are preserved, and the architecture is network independent, i.e. it supports various networks and multiple protocols on the same physical network. The limitation is the assumption that a host-capable machine exists, which is both connected to the interconnect and the Jini network.

4.5 The Anhinga Project

The goal of the Anhinga Project [Anhinga] is to develop variations of J2ME CLDC, Jini and JavaSpace technologies which will run in ad-hoc networks of small, mobile, wireless devices like PDAs, cell phones, pagers, and pocket PCs. Administrators of the project is representatives from Department of Computer Science at Rochester institute of Technology, an United States university located in Rochester, New York State.

The background for the Anhinga project is that mobile, wireless devices represents a potential for building distributed communication and collaboration applications that will operate on spontaneously formed (ad-hoc) peer-to-peer networks of the devices themselves, without requiring central servers or a fixed network infrastructure. But this potential remains unfulfilled due to lack of distributed middleware designed specifically for the small, mobile, wireless device environment. The project seek to develop an infrastructure to support distributed computing specifically to support collaborative applications running on wireless ad hoc networks of mobile computing devices. Components of the infrastructure are listed below.

- **Many-to-Many Protocol (M2MP)**: A simple protocol for communicating among a group of mobile devices in the vicinity in an ad-hoc wireless network. This will eliminate the complications of traditional packet routing by using broadcast messages.
- **J2ME CLDC extension**: Create an extension that adds lightweight security and dynamic class loading capabilities.
- **Anhinga Virtual Machine (AVM)**: A virtual machine implementation of the M2MP and J2ME CLDC extension, targeted to run on small, mobile, wireless devices.

- **MIDP extension**: An extension to J2ME MIDP to add lightweight APIs for reflection, object serialization, marshalled objects, and RMI. An API for the Many-to-Many Protocol will be supplied through the Generic Connection Framework.
- **Anhinga Device Profile (ADP)**: A profile implementation of the above specification targeted to run on mobile, wireless devices.
- **Jini Mobile Edition (JiniME)**: A variation of Jini Network Technology, built on the above infrastructure and providing federated service architecture for distributed applications running on mobile, wireless devices.
- **Anhinga Space**: A distributed tuple space service patterned after JavaSpace and built on the above infrastructure. It will provide tuple space shared among a group of mobile, wireless devices.
- **Proof of concept applications**: An application to demonstrate and study the above infrastructure, including chat, instant messaging, collaborative groupware, and others.

The current status of the project is that a Many-to-Many protocol and a RIT class file Library, which lets a program synthesize a class at run time, has been developed.

4.6 Summary

Jini is a distributed network technology, designed to handle dynamic environments. The technology is based on the assumption that Java is present in the network, and ensures a homogeneous environment by use of mobile code that is transferred between clients in the network. Three components constitute the Jini architecture, the infrastructure component, the programming model component and the service component.

Entities with abilities in the network are defined as services, whether they are software entities, hardware entities, or a combination. Services are registered with lookup services, and clients can request a lookup service for a particular service and receive a proxy representing the service as response. The proxy handles all communication with the remote service, and thus creates independence between the client and the transport protocol used. The only element that is discovered in Jini using a discovery protocol is the lookup service. These features are part of the infrastructure component.

The network ensures self-healing, i.e. that the community can adapt to changes, for example that entities join and leave without further notice. This is done through leases, that is to say that clients can uses a service for a predefined time before it has to revise the agreement, i.e. update the lease. Jini transaction model can ensure secure transactions between entities, and clients can ask to receive notification if something of interest becomes available, i.e. event notification. These features are part of the programming model component.

The service components are the entities that come together to form the Jini community, and are based upon the infrastructure component and use the programming model component. Example of services is the lookup service, the JavaSpace service and the transaction service.

Jini Surrogate Architecture makes resource constrained and non-java enabled device able to participate in the Jini community. A client can download a surrogate object to a surrogate host. The surrogate object represents the client on the Jini network, and the surrogate host acts on behalf of the client. As part of the surrogate architecture there is defined several interconnect protocols, for different transport protocols, that connect a client with the surrogate host.

Chapter 5

JXTA

"Putting things next to each other, which is really what peer-to-peer is about ... having groups of things come into affiliation ... into juxtaposition for a while and then move on."
Bill Joy

JXTA, short for Juxtapose, as in side by side, is an open-source project that defines a set of protocols for ad-hoc, pervasive, peer-to-peer computing. It started as a research project called Juxtapose at Sun Microsystem [Sun] in the summer of 2000, led by Chief Scientist Bill Joy. In April 2001 the project continued development in an open-source community, the JXTA community, where everyone who wishes may contribute to the further development.

JXTA strives to provide a base P2P infrastructure over which other P2P applications can be built, seeing that standards are absent in the P2P world. Current P2P solutions use different protocols, architectures and implementations. The base of JXTA consists of a set of protocols that are language independent, platform independent, and network agnostic, i.e. they do not assume anything about the underlying network. Each protocol serves a special purpose, and is used to exchange messages between the entities in the P2P network.

This chapter will give a brief introduction to JXTA, its goals, its terminology and concepts, the architecture and protocols. One of the projects in the JXTA community is to develop a JXTA implementation for J2ME enabled devices, and a description of this implementation will be provided.

5.1 Introduction to JXTA

Project JXTA envisions a world where each peer, independent of software and hardware platform, can benefit and profit from being connected to millions of other peers. The vision of JXTA, as stated in the vision statement, is quoted below.

"Project JXTA is building a core network computing technology to provide a set of simple, small, and flexible mechanisms that can support P2P computing on any platform, anywhere, and at any time. The project is first generalizing P2P functionality and then building core technology that addresses today's limitations on P2P computing. The focus is on creating basic mechanisms and leaving policy choices to application developers." [Krishnan]

The objectives of the JXTA Project are based on what is considered shortcomings in many peer-to-peer systems ([Comp2001], [Gong]):

- Interoperability,
- Platform independence, and
- Ubiquity

Interoperability in JXTA is achieved by designing the framework 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. Many peer-to-peer (P2P) systems are built for delivering a single type of services. Napster, for example, provides music file sharing, ICQ provides instant messaging and Gnutella provides generic file sharing. These systems are incompatible because of the lack of a common underlying P2P infrastructure. Each vendor creates their own P2P user community, duplicating effort in creating software and system primitives commonly used by all P2P systems. Developers that want to offer the same service to two different communities will have to develop the same service twice, or develop a bridge system between the communities.

JXTA technology is designed to be *platform independent*, i.e. it is independent of programming language, system platforms, and networking platforms. The API of many P2P systems is based on a particular operating system and a specific network protocol, and as a result, it is unlikely that two systems will interoperate.

Further, JXTA offers *ubiquity*. It is designed to be implemented on any device with a digital heartbeat, including sensors, consumer electronics, PDAs, appliances, network routers, desktop computers, datacentre servers, and storage systems. Designers of P2P often seek profit as fast as possible, by targeting the largest installed base of consumers, and in doing so choosing Microsoft Windows as target platform. A result might be that many dependencies on Wintel-specific (Windows-Intel) features are designed into the system. As it is very likely that other devices and systems will have great profit of P2P technology, betting on any particular segment of hardware or software system is not future proof.

JXTA has several advantages and limitations, some were found under the study of JXTA, others were pointed out in [Wilson]. The advantages are:

- Seeks to provide a standard way to communicate in a P2P network.
- Provides a far more abstract language for peer communication than previous specialized P2P protocols, enabling a wider variety of service, devices, and network transports to be used in P2P networks.
- JXTA does not dictate any particular programming language or environment.
- Extensible Markup Language (XML) or binary representation of data is used to exchange messages, both understood by a majority of the platforms currently available. XML provides a standards-based format for structured data that is well understood, well supported and easily adapted into s a human-readable format, making it easy for developers to debug and comprehend.
- Provids P2P functionality to “every device with a digital heartbeat”.
- JXTA is available for peers behind firewalls and NATs.

Limitations of JXTA:

- Many claim that the JXTA initiative to create a P2P standard comes to early, since peer-to-peer has not had time to mature.
- JXTA constitutes an extensive framework. This might prevent the use of the specification since there is plenty to learn before it can be used, and a lot to keep track of.
- JXTA does not attempt to address how services are invoked, with exception of core services. Several standards exist for defining a service invocation, such as the Web Service Description Language (WSDL), but none have been chosen specifically by the JXTA Protocol Specification. JXTA provides a generic framework for exchanging information between peers, so any mechanism, such as WSDL, could potentially be used via JXTA to exchange the information required to invoke services.
- The designers of JXTA infused a lot of flexibilities throughout the JXTA Protocols Specification, and some have criticised this flexibility. Though JXTA’s use of XML specifies all aspects of P2P communication for any generic P2P application, JXTA might not be suited to a specific standalone P2P application. In an individual application, the network overhead of XML messaging may be

more trouble than it's worth, especially if the application developer has no intention of taking advantages of JXTA's capabilities to incorporate other P2P services into their application.
- The platform's abstraction from the network transport has been criticised as a potential area of excess. If most P2P applications today rely on the IP protocol to provide a network transport, why does JXTA go to such lengths to avoid tying the protocols to a specific network transport? Why not specify IP as the assumed network transport and eliminate the overhead?
- JXTA protocols do not by themselves promise interoperability. Just because two applications are built on top of JXTA does not mean that they can magically interoperate. Developers must design applications to be interoperable. However, developers can use JXTA, which provides an interoperable base layer, to further increase interoperability.

Developers must balance flexibility with performance when implementing their P2P applications. JXTA provides the most well-rounded platform for producing P2P applications that have the flexibility required to grow in the future, but this does not indicate that it is the "best" or most efficient solution for implementing a particular P2P application. The core value of the JXTA platform is the capability to leverage other P2P services, and enable widespread development of P2P communities. While the technology is in its early stages today, it is expected to mature over time to provide a robust and reliable framework for P2P computing.

The JXTA community has discussed the possibility to submit the JXTA protocols to an outside organization and process for formal standardization, and most appropriate forum to standardize JXTA protocols was considered IETF. Late June 2002 it was announced that the JXTA protocols has been accepted as an "Internet-Draft" [JXTAdraft] and has started the way to standardization.

The rest of this chapter will focus on the JXTA specification and the reference implementation of the specification, which is written in Java and imposes several dependencies. For example are the JXTA protocols implemented to support HTTP and TCP/IP transport.

5.2 Architecture

The JXTA architecture consists of three layers; the platform layer, the service layer and the application layer as shown in Figure 5-1.

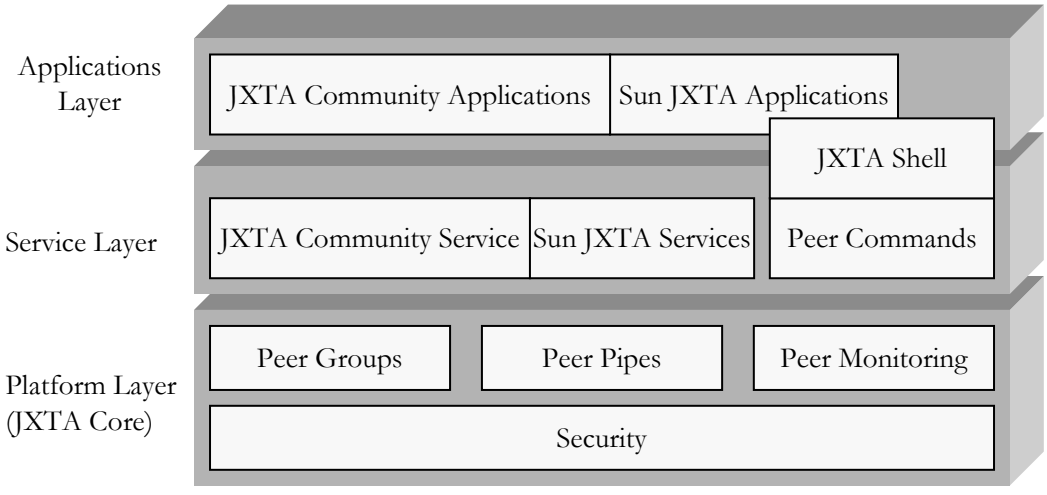


Figure 5-1 The JXTA 3-layer architecture

The boundary between services and applications is not rigid. An application to one customer can be viewed as a service to another customer. The entire system is designed to be modular allowing developers to pick and choose a collection of services and applications that suites their needs.

5.2.1 The platform layer

The platform layer is also known as JXTA core layer, and provides the absolutely essential elements to every P2P solution. The elements of the core layer are listed below, and are ideally shared by all P2P solutions.

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

The first six elements is discussed in section 5.3, while the protocols are covered in section 5.4. Note that JXTA's six main protocols are implemented as services, but located in the Platform Layer and designated as 'core services' to distinguish them from the service solutions of the Service layer. JXTA core layer is the fundamental core of the JXTA solution, and all other aspects of a JXTA P2P solution in the Service or Applications layer build on this layer to provide functionality.

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. Communication across a firewall or NAT (Network Address Translation) is solved in two ways in the reference implementation. Either at least one peer in the peer group inside the firewall must be aware of at least one peer outside of the firewall, and both parties must support HTTP transport. In addition the firewall must allow HTTP data transfers, but the HTTP transfer need not be restricted to port 80. The other solution is that the JXTA node can be configured to communicate via a HTTP proxy server, and in this setting the above mentioned peers become routers for P2P traffic among the other peers in the peer group. Multiple peers can route P2P JXTA messages across the firewall. Today, the setting to support firewalls and HTTP transport is done in configuration files, but more advanced treatments are expected to come. [JXTAfaq]

5.2.2 The service layer

The service layer includes network services, both common and desirable in the P2P environment, but may not be absolutely necessary for a P2P network to operate. Example of services are searching for resources and peers, sharing of documents between peers, peer authentication, and Public Key Infrastructure (PKI) services.

Services on the JXTA network is published, discovered and invoked by cooperation and communication between peers. Two levels of services are recognized by the JXTA protocols:

- Peer Service
- Peer Group Service

A peer service is accessible only on the peer that is publishing the service, and if that peer fails, the service also fails. Services provided by a peer group is composed of a collection of instances of the service running on multiple members of the peer group. As a result, the collective peer group service is not affected if a peer fails since it is available from another peer member.

Services might be built by the JXTA community or by the Project JXTA team. Services built on top of the JXTA platform provide specific capabilities that are required by a variety of P2P applications and can be combined to form a complete P2P solution.

5.2.3 The application layer

The application layer provides common P2P applications like instant messaging, by building on the capabilities of the service layer. Sometimes it is difficult to determine what constitutes an application, and what constitute a service since an application may encompass only a single service, or aggregate several services. An application is usually indicated by the presence of user interface. One example is the JXTA Shell, an application built on top of the Peer Commands, which is a service. As a result, the JXTA Shell is spread across the Application/Service boundary

Applications in JXTA include those built by the JXTA community and the Project JXTA team, which mostly contribute with demonstration applications like the JXTA shell.

5.3 Terminology and concepts

JXTA specification defines a number of concepts that are common for P2P networks, and as a result are the primary components of the JXTA platform. This section will give an overview of these concepts.

5.3.1 Peer

Peers are the nodes in a P2P network, and the fundamental processing unit of any P2P solution. They can manifest in the form of a processor, a process, a machine, or a user, and can include sensors, phones, PDAs, as well as PCs, servers, and supercomputers. They can also be an application distributed over several machines, and the connected devices like PDAs might not be directly connected to the network, but for instance via a synching cradle. Peers are capable of communicating using the protocols required by a peer, but does not need to understand all the JXTA protocols (see section 5.4). A peer can still perform at a reduced level if it does not support a protocol. ([JXTASpec], [Wilson])

Each peer performs some useful work, but the work is dependent upon the type of peer. There exists today three types of peers, and they define a set of responsibilities for the peer in relation to the P2P network as a whole. The different types of peers are [Wilson]:

- **Simple peers**: A simple peer is the simplest type of peer on the P2P network, and provides and consumes resources from other peers on the network. It is not responsible for forwarding messages on behalf of other peers, or providing third-party information to the network.
- **Rendezvous peers**: Provides simple peers with a way of discovering other peers and advertisements on the P2P network.
- **Router peers:** Router peers provide routing services to enable peers inside private internal networks behind firewall and NAT equipment to participate in a P2P network.

5.3.2 Peer Groups

Peer Groups is a collection of peers that have a common set of interests [JXTASpec]. Peers self-organize into peer groups, and each peer group have a unique identification. The JXTA protocols describe how a peer may publish, discover, join, and monitor peer groups.

The background for introducing peer groups was to divide the network space, a need that arose when all peer were able to communicate using the same protocol. Earlier P2P solutions were specialized and proprietary, and their associated protocols divided the usage of the network space according to application, i.e. for file sharing you used Gnutella, for instant messaging, ICQ. Peer groups divide the P2P network into groups of peers with common goals, which are based on the elements in the list below. [Wilson]

- **Scoping environment**: Peers typically forms and self-organize based on the mutual interest of the peers, and groups allow the establishment of a local domain of specialization. Peer groups can for example be divided based on application they wish to collaborate on as a group, or because they want to exchange services only among the members of the group, and not the entire P2P network.

- **Secure environment**: Groups create a local domain of control in which a specific security policy can be enforced. It limits the access to peer group resources by forming logical regions without respect to actual physical network boundaries.
- **Monitoring environment**: Peer groups permit peers to monitor a set of peers for any special purpose, for instance heartbeat, traffic introspection, or accountability.

The specification does not dictate when, where, or why to create a peer group, or the type of the group, or the membership of the group, and does not limit the how many groups a peer can belong to, or if nested groups can be formed.

A peer group provides a set of services where the core services are specified by JXTA, but additional peer group services can be developed for delivering specific services. The core services are:

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

The services are implemented in one or more of the JXTA protocols, or none, and peer groups are not obliged to implement all these services.

5.3.3 Network Transport

Exchange of data between peers is made possible by the network transport layer, a mechanisms to handle the transmission of data over the network. In JXTA the concept of network transport is broken down in three parts; endpoints, pipes and messages.

Endpoints

Endpoints are network interfaces used to send and receive data, i.e. the initial source or final destination of any piece of data being transmitted over the network.

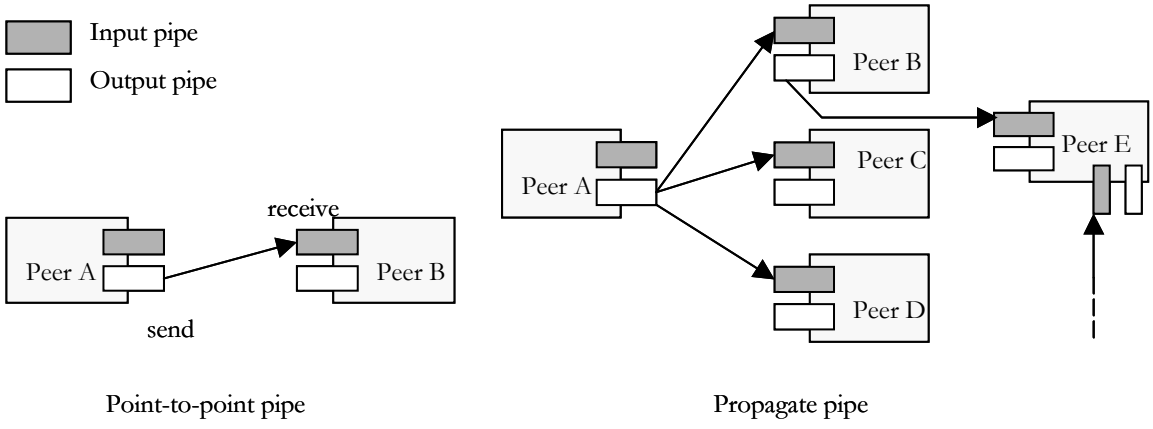


Figure 5-2 Point-to-point and propagate pipes

Pipes

Pipes are virtual communication channels that are used by peers to send and receive messages between services, or applications, over endpoints. The term virtual refers to the fact that peers do not need to

know their actual network addresses to use them. Pipes are uni-directional and asynchronous, connects two or more endpoints, and offers two modes of communication:

- **Point-to-Point Pipe**: Connects exactly two pipe endpoints, an input pipe end that receives messages sent from the output pipe end.
- **Propagate Pipe**: Connects one output pipe to multiple input pipes, and messages are sent to all listening input pipe ends in the peer group.

The two communication modes are visualized in Figure 5-2. Bi-directional, reliable and secure pipe services have been implemented on top of the core pipe services.

Messages

Communication in the JXTA environment is done by sending and receiving messages. The messages are containers of the data being transmitted over a pipe from one endpoint to another. To make JXTA interoperable a standard format is needed for the messages. JXTA uses messages in XML-format, but both XML and binary payloads can be sent. The messages are defined as an ordered sequence of named and typed contents called elements. As a result of the choice between XML and binary, each JXTA transport can use the most appropriate format for moving data.

5.3.4 Advertisements

All network resources, like peers, peer groups, pipes, endpoints, services and content, can be described by advertisements. Advertisements are language-neutral metadata structures that describe the network resources, i.e. they dictate the structure and representation of the data. Project JXTA standardizes a set of advertisements, and developers are free to subtype these advertisements to create their own types. The core advertisements are [JXTASpec]:

- Peer advertisement
- Peer Group advertisement
- Pipe advertisement
- Module advertisement
- Peer Info advertisement
- Content advertisement
- Peer Endpoint advertisement

Advertisements are cached, published and exchanged between peers to discover and find available resources. Peers discover resources by searching for their corresponding advertisements, and each advertisements has a lifetime that specifies the availability of the associated resource in the network. The reference implementation provides three ways for peers to discover an advertisement [Wilson]:

- **No discovery**: The peer relies on a cache of previously discovered advertisements to provide information on peer resources.
- **Direct discovery**: Peers on the same LAN may be capable to discover each other directly without relying on an intermediate rendezvous peer by use of the broadcast or multicasting capabilities of their native network transport.
- **Indirect discovery**: Requires use of a rendezvous peer to act as a source of known peers and advertisements, and to perform discovery on a peer's behalf.

5.3.5 Entity Naming

To uniquely identify items on a P2P network, like peers, peer groups, pipes and contents, some information to represent the uniqueness is needed. A peer using TCP/IP transport could be identified by its IP

address, but use of system-dependent representation is inflexible and cannot provide a system of identification that is independent of the operating system or network transport.

Any device, regardless of their operating system or network transport, should be able to participate in an ideal P2P network. As a result a system-independent entity-naming scheme is a requirement for a flexible P2P network. A JXTA ID uniquely identifies an entity and serves as a canonical way of referring to that entity. IDs are normally presented as URNs, which are a form of URIs that "... are intended to serve as persistent, location-independent, resource identifiers"[JXTASpec]. JXTA IDs are presented as text.

5.3.6 Security

Security is not implicit in the JXTA framework, but the framework provides the means for strong security to be implemented using well-understood, trusted approaches. The XML messages allow adding a large variety of metadata information to messages such as credentials, digests, certificates, public keys and similar. Credentials are a token used to identify the sender of the messages, and can be used to establish public/private key and digital certificate technologies to implement an effective approach to authentication.

Project JXTA have made three choices when it comes to security ([JXTASec], [ITProf]):

- Adoption of Transport Layer Security (TLS)[TLS].
- End-to-end transport independence of JXTA protocols.
- Use of X509.V3 digital certificates, where centralized certificates authorities not are mandatory, but not excluded either.

Transport Layer Security (TLS) is an industry protocol for secure transport of information, and a continuation of Secure Socket Layer (SSL). TLS is currently under development by the Internet Engineering Task Force (IETF) Network Working Group. The implementation of TLS chosen by the JXTA project to be included in the JXTA distribution is Pure TLS [PureTLS] from Claymore Systems.
.

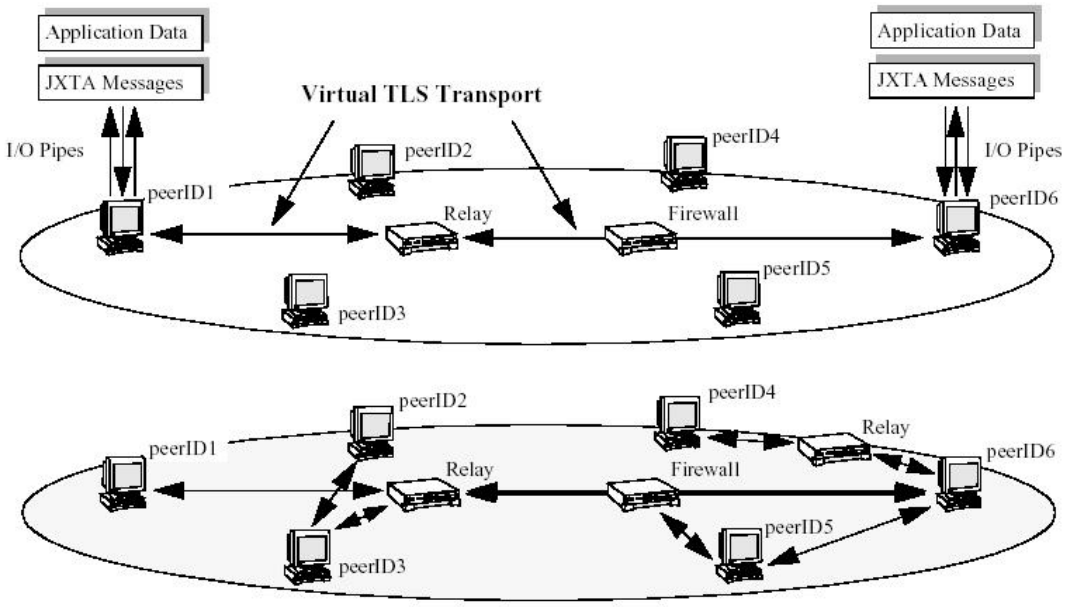


Figure 5-3 Project JXTA Virtual TLS transport [JXTASec]

Figure 5-3 visualizes the JXTA TLS transport. On the physical level, peers are connected through firewalls and designated relays. TLS connections create a virtual layer on top of the physical layer where JXTA messages can be exchanged securely

End-to-end transport independence was one of the design choices of project JXTA. JXTA is often compared to TCP/IP that links Internet nodes together where JXTA technology connects peer nodes with each other. Different transport standards can be used to transport the JXTA messages, like TCP/IP, with no impact on the JXTA messages since these have pre-defined format, and may include multiple data fields. The importance of this is that encrypted content will always remain encrypted, regardless of protocol conversions that may occur between networks.

Digital certificates are used in JXTA to ensure the identities of parties to a transaction, and fulfil the requirements to authentication and non-repudiation. Digital certificates are issued by a trusted-party, i.e. a X509.V3-compliant certificate authority (CA). But a centralized CA is not always appropriate in peer-to-peer computing, since the involved parties want to be able to conduct a secure transaction without involvement of a centralized framework. The JXTA solution is to allow peers to become their own certificate authorities, generating their own root certificate that verifies that they are associated with a specific public key. JXTA security model allows peers form peer groups where a member can be designated as the CA for that group. But a peer group can also use a well-known certificate authority to issue the certificates.

5.4 The Protocols

Protocols are needed at every data exchange to dictate what data gets sent, and in what order it gets sent. They structure the exchange of information between the participants using rules agreed upon by all parties. The protocols in P2P are needed to define every type of interaction a peer may perform as part of the P2P network. This includes:

- Finding peers on the network
- Finding what services a peer provides
- Obtaining status information from a peer
- Invoking a service on a peer.
- Creating, joining, and leaving peer groups.
- Creating data connections to peers.
- Routing messages for other peers.

Advertisements simplify the protocols required to make P2P work since they dictates the structure and representation of the data and only leaves the protocol with the organization of exchanging advertisements. The six protocols in JXTA defined today are:

- Peer Discovery Protocol (PDP)
- Peer Information Protocol (PIP)
- Peer Resolver Protocol (PRP)
- Pipe Binding Protocol (PBP)
- Endpoint Routing Protocol (ERP) / Peer Endpoint Protocol (PEP)
- Rendezvous Protocol (RVP)

All protocols were designed to be easy to implement on unidirectional links and asymmetric transports. The intention was for the protocols to be as pervasive as possible, and easy to implement on any transport. JXTA permits any unidirectional link to be used when necessary, and this might improve the overall network connectivity in the system. Efficient bi-directional communication can be realized by implementing the protocols on reliable and bi-directional transports such as TCP/IP or HTTP. [JXTAvn]

Figure 5-4 shows communication between to peers using a protocol, and also how the layers of the protocol stack build on top of and rely on the layers below them.

Realizing that to provide a universal base protocol layer it must adopt a suitable representation that a majority of the platform currently available can understand, XML was chosen to define all protocols in JX-

TA. But JXTA does not depend on XML encoding; it is just a convenient form of data representation. Protocol can also use binary messages, which is more suitable for devices with no XML parser.

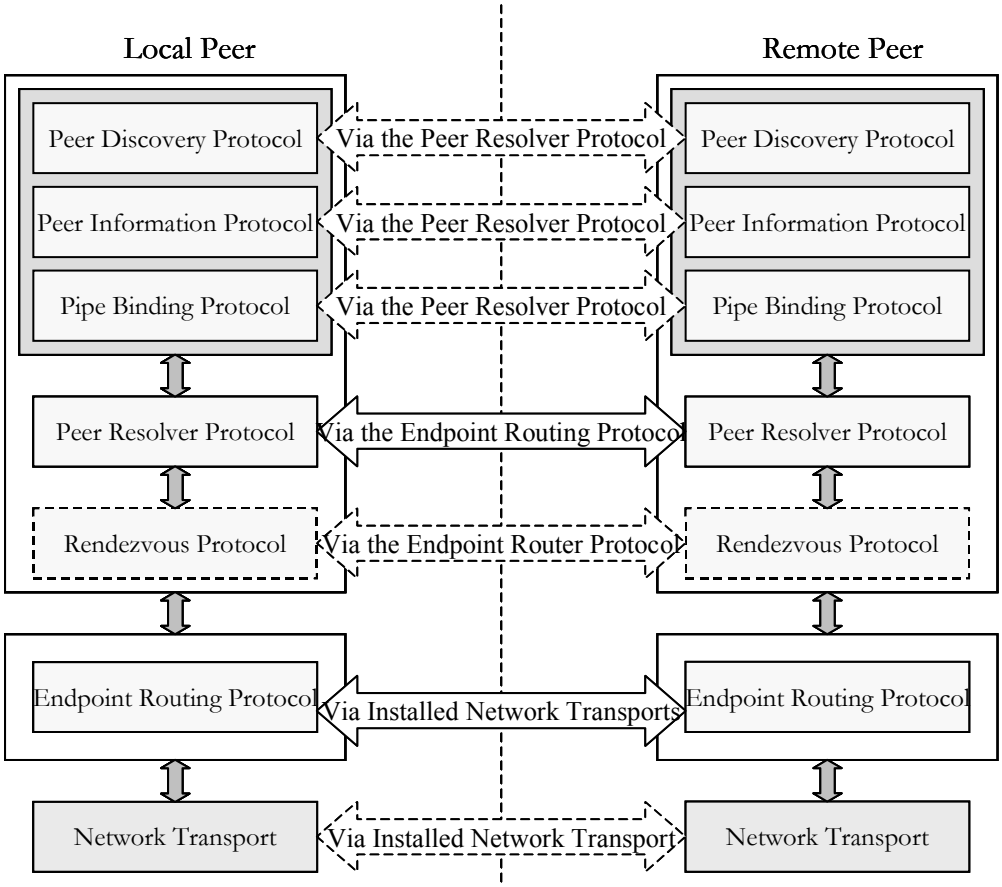


Figure 5-4 The JXTA Protocol Stack

The responsibility of each of the protocols is described in the next subsections. Note that a peer only need to implement the protocols that is requires. For instance if a device stores all advertisements it uses in its memory, the peer has no need to implement the Endpoint Routing Protocol. Or if it has no desire to obtain, or provide, status information to other peers, it may choose not to implement the Peer Information Protocol. [JXTAvn]

5.4.1 Peer Discovery Protocol

Peer Discovery Protocol (PDP) is used by peers to discover all published JXTA resources, and for the peer to advertise its own resources. Since every peer resource is described and published using an advertisement, the protocol helps a peer to discover an advertisement. The protocol can be extended to support more superior discovery mechanism, but the inclusion of this default protocol ensures that JXTA peers can understand each other at the very basic level.

5.4.2 Peer Information Protocol

Peer uses the Peer Information Protocol (PIP) to obtain status information about other peers, for instance state, uptime, traffic load and capabilities. Receiving a PIP message leaves the peer with several options. It can ignore the ping, send a simple acknowledgement (consisting only of its uptime), or send a full response, which includes its advertisement.

5.4.3 Peer Resolver Protocol

Peer Resolver Protocol enables a peer to send a generic query to search for peers, peer groups, pipes, or other information to one or more peers, and receive response (or multiple responses) to the query. In other words the protocol implements a query/response protocol, where a query to a peer only can be sent after the peer is discovered via PDP. The intention of the protocol is to standardize the format of these queries. The response message is matched to the query via a unique id included in the message body. PRP is typically implemented only by those peers that have access to data repositories and offer advanced search capabilities.

5.4.4 Pipe Binding Protocol

The Pipe Binding Protocol (PBP) allow a peer to establish a virtual communication channel or pipe between one or more peers, by binding two or more pipe ends of the connection (input and output pipe) to a physical peer endpoint. PBP is used by the peer to create a new pipe, bind to an existing pipe, and unbind from a pipe at runtime, and use PRP for sending and propagating pipe binding requests.

5.4.5 Endpoint Routing Protocol

A peer uses the Endpoint Routing Protocol (ERP) to discovery a route for a message to a destination peer. Two peers might not be directly connected to each other, as a result of different network transport protocols, or separation by firewall or NATs. ERP is used to determine the routing information by asking peer routers for available routes. The peer itself becomes a peer router by implementing the Endpoint Routing Protocol.

Endpoint Routing Protocol is also known under the name Peer Endpoint Protocol (PEP).

5.4.6 Rendezvous Protocol

The Rendezvous Protocol (RVP) allows a peer to subscribe or be a subscriber to a propagation service. In a peer group, peers can be rendezvous peers, or peers listening to rendezvous peers. The RVP allows messages to be sent to all of the listeners of the service. In order to propagate messages, the RVP uses the PRP.

An earlier version of this protocol is called Peer Membership Protocol (PMP), and is a more specialized protocol intended for manage group membership, for example subscribing, obtaining information and cancel a membership.

5.5 JXTA community

JXTA started as a research project at Sun Microsystems, but upon recognizing that the effort would benefit from expert coders outside of Sun, the JXTA community was formed and the JXTA web site [JXTAws] was opened in April 2001. The JXTA community allow all developers to join in the effort of creating JXTA, and all specifications and implementation code is open for developers under the Apache Software License.

Participating members of the JXTA Community use the JXTA web site to provide papers on the technology, tutorials, and hosts JXTA projects. These projects can be initiated by JXTA community members, after approval by the community, and anyone that wants is welcome to join. Projects can be divided in six broad categories; core components, services, applications, demos and tutorials, forge (early-stage JXTA projects) and other.

The core components projects include the platform project, the reference implementation of JXTA in Java, and implementation of JXTA in other languages, like C, Perl, Smalltalk, and Phython. Other projects are more specialized for specific devices, for example JXTA for Pocket PC and a TINI binding.

The most interesting project in connection with this report it the JXTA for J2ME (JXME) project [JXME].

5.6 The JXTA for J2ME project

JXTA for J2ME [JXME], also known as JXME, is an implementation of JXTA to enable embedded devices to participate in the JXTA community using J2ME, and communicate with JXTA peers running on desktops, workstations or servers. In this way JXTA for J2ME extends the vision that JXTA should work on any device.

The vision of JXTA for J2ME is to provide an infrastructure for self-organizing networks for devices. This will allow for the developing of applications for spontaneous, ad-hoc, collaboration. One possible scenario is that one could spontaneously setup a Personal Area Network (PAN) of devices. Thus, when a PDA comes near a cell phone and/or a laptop, they could talk to each other.

When designing JXME, several constraints imposed by cell phones and similar devices had to be taken into consideration and influenced the design choices. Today's phones have an average total limit of about 123K for storing all MIDlet suites, and some manufacturers have even less space. In addition, the size of persistent storage and heap is limited, and the device may have high latency bandwidth.

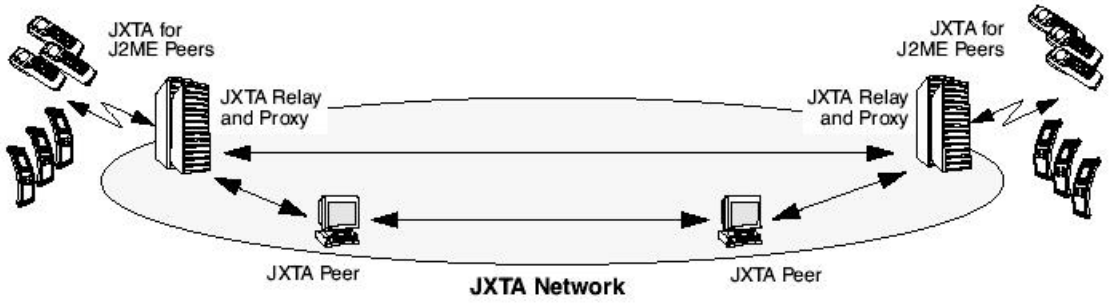


Figure 5-5 JXTA for J2ME [JXMEwp]

A requirement is that the devices use the Connected Limited Device Configuration (CLDC) and the Mobile Information Device Profile (MIDP). This limited implementation was necessary to JXTA-enable these devices since the full JXTA implementation requires resources that exceeds the resources of embedded devices. An implementation for CDC is also under development, and according to one of the project owners it runs fine with Personal Profile.

Current implementations of the J2ME platform, specifically CLDC 1.0 and MIDP 1.0, have several constraints, and to provide true peer-to-peer communication a relay is needed. MIDP 1.0 does have limited libraries, lack of an XML parser, support for outgoing HTTP only, and no security support. These limitations are reflected in the JXTA for J2ME's capabilities. As a result, JXME is implemented using HTTP transport. For messages, JXME uses binary messages that conform to JXTA's Binary Message Format, and security is not supported. Other features than security that are unsupported are relay service, router service, and rendezvous service. These features will be supported when cell phones mature and have better support for them, and MIDP have available libraries to support them.

The intention for the future is to provide for true peer-to-peer communication, without the need of a relay. MIDP 2.0 will provide for such an opportunity, since it includes further network protocol support. As of today, to accomplish the goals while considering the constraints in using J2ME, a relay server is needed to perform some of the work. In addition this relay will make JXTA available for devices behind a NAT. JXME is therefore split in to modules; a Peer and a Relay Service.

5.6.1 JXTA for J2ME Peer

The JXTA for J2ME Peer will provide the building blocks necessary to communicate with the JXTA for J2ME relay, and indirectly with JXTA. The first version of the API was presented in late March 2002, and is under constant change, which should have minimum to no impact on the applications developed.

JXME API available consists of three classes; *PeerNetwork*, *Message* and *Element*. Figure 5-6 shows the relationship between the classes. The PeerNetwork class is used to communicate with the JXTA network, and uses Message to create messages to send to other peers. A Message consists of one or more Elements describing attributes in name-value pairs. The API is implemented in a way that the underlying transport protocol could easily be changed as long as J2ME's Generic Connection Framework, defined in CLDC, supports the new network technology.

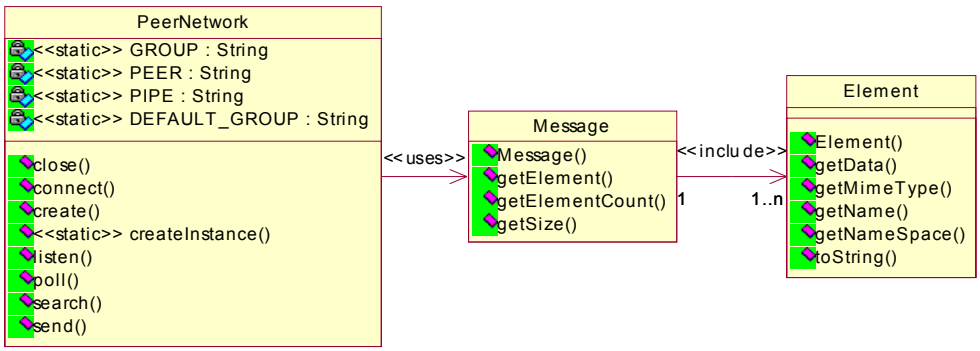


Figure 5-6 JXTA for J2ME API

The first version of JXME supports the following features:

- Discovery of pipes, groups and peers
- Able to create pipes, both point-to-point and propagate pipes, and groups.
- Join groups
- Communicate with other JXTA users through JXTA pipes.

5.6.2 JXTA for J2ME Relay Service

JXTA for J2ME peers can only acts as edge, or simple, peers, i.e. they cannot assume the role of more sophisticated peers that offer services to other members in the peer group. The peers themselves can perform only basic tasks, and heavier operations have to be done by peers called JXTA relays, as shown in Figure 5-5.

The relay is part of the Java 2 Standard Edition (J2SE) version of JXTA [JavaJXTA], and runs on a JXTA Rendezvous peer. The relay service is responsible for the following:

- **Provide interoperability with JXTA protocol**
- **Represent the JXTA for J2ME peer in the JXTA network**: The relay should act as a proxy for the JXTA for J2ME peer, and perform user, group, and peer discovery. Other tasks are to create pipes and groups, join groups, and communicate. The relay will also store all incoming messages for a JXTA for J2ME peer, and the J2ME peer will periodically poll the relay to get its incoming messages.
- **Filter JXTA traffic**: The relay will filter all incoming queries from the network and respond on behalf of the wireless peer.

- **Trim and optimize advertisements**: The relay should filter unnecessary advertisements since a JXTA for J2ME peer does not have enough memory to store all incoming advertisements. Messages and incoming advertisements are stripped down to a minimum, so that they will be small.
- **Translate Messages**: JXTA for J2ME peers uses binary messages, and the relay service will have to translate JXTA XML messages into binary messages and vice versa.

Relays are not a new idea for JXTA. They are already used to access JXTA peers behind firewalls and allow networks using NAT to communicate with the outside world. JXTA peers do not have to maintain a static relationship with a designated JXTA relay, and is therefore unlike the client-server model. JXTA peers can change relays dynamically, or be in relationship to multiple relays at the same time. A future scenario is that peers can search for JXTA relays and configure one of them to be its default.

5.7 Summary

JXTA is a new distributed platform designed to solve a number of problems in modern distributed computing, especially in the area broadly referred to as peer-to-peer (P2P) computing. The objective is to provide interoperability between entities in the network, be platform independent and offers ubiquity, i.e. any device with a digital heartbeat can participate.

The JXTA specification defines a number of concepts that are common for P2P networks, and as a result is the primary components of the JXTA platform. The concepts are peers, peer groups, endpoint, pipes, messages, advertisements, entity naming and security. A peer is a node in a P2P network, while a peer group is a self-organized collection of peers that have a common set of interests. Communication between two peers, in the form of messages, goes through pipes, from one endpoint to the other. Advertisements are language-neutral metadata structures that describe the network resources, i.e. they dictate the structure and representation of the data. They are published and exchanged between peers to discover and find available resources, and have a lifetime that specifies the availability of the associated resource. Entities in the network are identified by a unique ID that are independent of network and other system specific notation. Security is provided through use of Transport Layer Security (TLS) and digital certificates, and in addition the transport independent design provides end-to-end transport security.

The base of the JXTA specification consist of a set of protocols that are language independent, platform independent and network agnostic, i.e. they do not assume anything about the underlying network. Six protocols are already defined. The protocols enable discovery of resources on the network, the possibility to obtain status information about other peers, to bind a pipe to a physical peer endpoint, find an appropriate route to another peer, subscribe to a propagation service, and a protocol to perform generic request/response queries. The protocol uses XML or binary messages to transfer information between peers.

JXTA for J2ME, JXME, is a project that aims at providing JXTA functionality to java-enabled, embedded, wireless devices. As a result of the network limitations in MIDP, the implementation is dependent upon a relay to do most of the work. The other part of the implementation, the JXTA for J2ME peer, provides a simple API that uses HTTP to communicate with the relay.

Chapter 6

Related work

"Variety's the very spice of life,
that gives it all its flavor."
William Cowper

Different projects are initialized and focuses on technologies that are related directly or indirectly to the network technologies discussed in the former chapters. This chapter takes a brief overview of some of them and their functionality according to java-enabled mobile devices. The project presented is the AMIGOS project at NTNU, which includes different technologies to form Meeting Places for users to establish and participate in. Other interesting new technologies are Web Services and the .NET technology from Microsoft, which both have some of the same goals as JXTA and Jini.

6.1 The AMIGOS project

AVANTEL (Advanced Telecom services) [Avantel] is a project co-operation between NTNU, Telenor and Ericsson established in early 2001, and is the starting point of the AMIGOS project. AVANTEL is related to and support the intentions and visions of PATS (Program for Advanced Telecom Services) [Pats], a research program agreement between NTNU, Compaq Computers Norway, Ericsson and Telenor, which vision is "to establish [...] a virtual centre of excellence on advanced heterogeneous services and fast service development" [Avantel].

The focus of AVANTEL is to utilise platforms and development environments in combination with mobile terminals, and demonstrate the new range of possible telecom services available with introduction of new technologies. These technologies is among others GPRS and UMTS, and new software development libraries (APIs) for service logic, like Parlay/OSA, which is a set of open, standardized APIs to access core network functionality.

AMIGOS (Advanced Multimedia In Group Organized Services) [Amigos] is a service platform under development in the AVANTEL project. Further, it is an example service that aims at giving the activities in the AVANTEL project a unifying direction. The AMIGOS service is a generic service that provides users with the possibility to establish and participate in Meeting Places, i.e. virtual rooms where interactions optionally take place between the participants [Amigos02]. A Meeting Place might be limited to geographical areas, and the interaction is exchange, both received and possibly sent, of computer based information and media streams like text, audio and graphics. One of the key points is that the users of the service can themselves establish instances of the generic service, i.e. the Meeting Place, without involving the network operator ([Amigos01], [Amigos03]).

Users should be able to get to know the valid and established Meeting Places. The visible Meeting Places can be dependent on context information as geographical position or information from the user's profile, and is determined by the generic browsing service. Users can join Meeting Places, and participate in several at the same time, but might be blocked from joining according to features set by the owner of the

Meeting Place or charged for the participation. Meeting Places are designed with great flexibility regarding what actually happen in the Meeting Place, and configuration of the actual behaviour should be easy for the initiating user [Amigos03]. Example of a Meeting Places is *Background Music* where party music is chosen based on the musical taste of the users present at the party, or contribution of media files from the guests. Others are *Community Wall*, a electronic bulletin board for sharing of information to a community of users, and *Mega Chat* that is an entertainment service providing conferencing among participants and works a lot like chat, with the exception that it are able to translate text to speech [Amigos05].

As a result of the requirements specified, the generic functions in AMIGOS is logon, log off, browsing Meeting Places, and establishing, participating in and leaving a Meeting Place [Amigos04]. The modeling and implementation of the important service execution components in AMIGOS are intendend to be done using ServiceFrame, a framework from Ericsson. ServiceFrame has as objective to "address the principal underlying problems seeking to provide sound and viable solutions that enable rapid development of advanced, hybrid and personalized services without sacrificing the quality" [SFrame02]. The thought structure of the AMIGOS system is shown in Figure 6-1.

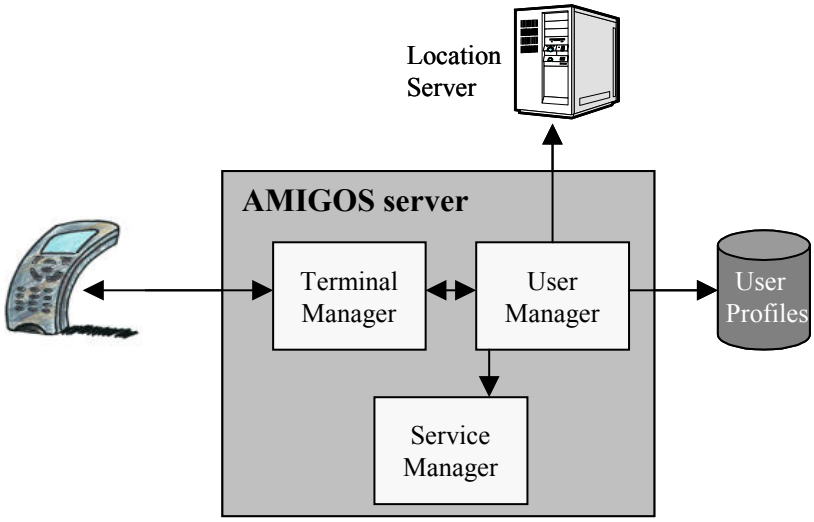


Figure 6-1 The AMIGOS Architecture

The AMIGOS Server will be implemented using ServiceFrame, where three of the main components are the Terminal Agent, the User Agent and the Service Manager. Upon connecting to the server, a client request is handled by the Terminal Agent which forward the request to the User Agent for authentication. The User Agent is responsible for all information about the user, which are stored as user profiles in a central repository, and authenticate the client by comparing the submitted credentials agains the credentials in the user profile database. A Meeting Place Service Manager is responsible for keeping track of the available and valid meeting places in the AMIGOS environment, that can be situated anywhere on the network. Meeting Places shown to a user may depend upon the users location, and the location is retrieved from a Location Server. The Location Server is situated outside ServiceFrame, and compute the location of a device based on information from the device, from the network, or information from both client and network. As a result of the placement of the location server it will be accessible for any device on the network. [Brandis]

Terminals covered are GSM mobile phones, hand held computers (PDAs), portable computers and ordinary telephone (PSTN/ISDN), and the connections that should be supported is 9.6 kbps GSM data, GPRS, Bluetooth, WLAN and LAN. In addition it should support a variety in operating systems, like Windows, Linux and Microsoft Pocket PC and the J2ME environment.

6.2 Web Services

The goal of Web Services is to make application functionality available over the Internet in a standardized, programmatic manner. Application should be able to communicate regardless of the implementation language, platform they were developed for, and the object models and internal protocols they use. [Frisk]

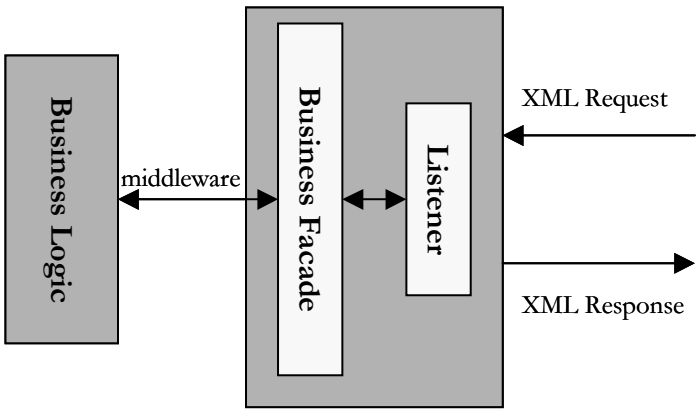


Figure 6-2 Generic Web Service Architecture

Web Service features are based on the combined strength of different middleware, like RMI, Jini, CORBA and DCOM, in addition to the strength of the Web, simple access and ubiquity. Services are effectively implemented in a traditional middleware platform, and the Web complements the platforms by providing a uniform and widely accessible interface and access. The generic Web Service architecture is shown in Figure 6-2. A listener handles the access, while a facade expose the operations supported by the business logic. The logic is implemented by a traditional middleware platform. [Vasu]

6.2.1 Web Service Platform

The Web Service Architecture stack varies from one vendor or organization to another. Some of the most popular Web Service architecture stacks developed the last few years are the World Wide Web Consortium's (W3C's) stack [W3Cws], IBM's stack and Microsoft's stack. They are all based on some technologies that form the common ground, called the Web Service Platform. These technologies have been discussed in connection with Web Services since the technology's early beginning, and are summarized below:

- Extensible Markup Language (XML)
- Simple Object Access Protocol (SOAP)
- Web Service Definition Language (WSDL)
- Universal Description, Discovery and Integration (UDDI)

The base specifications use an underlying network to transport the data. Because of the use of XML and SOAP, Web Services are independent of the transport protocol used, but most implementations of Web Services use TCP/IP and/or HTTP.

The use of *Extensible Markup Language* (XML) and *Simple Object Access Protocol* (SOAP) [SOAP] simplifies the integration and interoperability problems between companies or other that want to exchange data. Extensible Markup Language (XML)[XML] is a markup language for documents containing structured information. The adoption of XML is based on the need for a technology to enable usage of richly structured documents on the web. Simple Object Access Protocol (SOAP) is a framework built on XML that allows a program to invoke service interfaces across the Internet without the need to share a common programming language or distributed object infrastructure. While XML provides a cross-platform ap-

proach to data encoding and formatting, SOAP defines a simple way to package information for exchange across system boundaries.

Web Service Definition Language (WSDL) [WSDL] describes operational information of Web Services, i.e. where the service is located and what the service does. It also describes in what manner a requester can make a connection to a Web Service. WSDL defines an XML grammar for describing this kind of information. Another technology that takes care of service descriptions is the *Universal Description, Discovery and Integration* (UDDI) [UDDI]. The UDDI database stores and distributes references about the Web Services different companies offers, but the most important characteristic for UDDI is that it defines a next-layer-up that lets two companies share a way to query each other's capabilities and to describe their own capabilities. The UDDI specification consists of many documents in addition to an XML schema that defines a SOAP-based protocol used to discover and register Web Services.

6.2.2 Web Service Models

Different models can be used for the Web Service architecture, but the primary model is the Service Oriented Architecture (SOA). The model is a distributed approach that assigns different roles to the involved parties in a Web Service, while abstracting all details in what manner a service is actually implemented or build. The description of the physical parts that must be included to build a working system is left to the SOA component model, which will not be discussed further. Other models are under development that is modelled for highly distributed environments [Frisk].

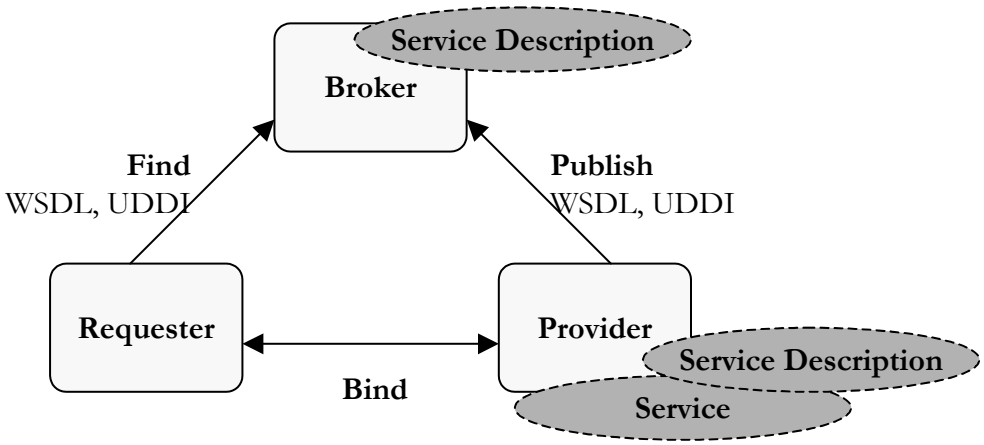


Figure 6-3 Service Oriented Architecture (SOA)

The SOA is the fundamental architecture for Web Services, and is illustrated in Figure 6-3. SOA defines three roles, *Requester*, *Provider* and *Broker*. The requester is the point of initiation for a request, while a provider is the point of request consumption, and hosts the exposed service descriptions. The broker is a repository, which keeps service descriptions, and allows a requester to find the services it wants to use.

The primary functions in SOA are explained in the list below.

- **Publish:** Lets a provider advertise its service interfaces to a broker
- **Find:** Lets a requestor discover a service description interface in the broker and retrieve it. The requestor uses the description to create a proxy interface, usually a SOAP-proxy.
- **Bind:** The bind function provides the requestor with the ability to bind to a provider, and consume its available services.

6.2.3 Web Services and J2ME

Access to Web Services from java-enabled, mobile devices should not be hard to accomplish. The only requirement is the ability to handle XML and SOAP, and there exists two project exploring both XML and SOAP for J2ME, called kXML [kXML] and kSOAP [kSOAP] respectively. Other XML parsers also exist, like TinyXML and NanoXML, but these were both ported to the KVM in contrast to the kXML that was specifically designed for CLDC. What might be the problem is the resource consumption of a XML parser since J2ME MIDP enabled devices already has minimal resources.

Sun has taken an initiative to define an optional package to provide standard access for J2ME to Web Services [JSR-172]. The specification request was accepted in the Java Community Process (JCP) in first quarter of 2002, and anticipated schedule is to release it in the summer of 2003. APIs that are intended to be included are:

- APIs for basic manipulation of structured XML data, i.e. parsing. The package will be a subset of JSR 63 JAXP.
- APIs for conventions for enabling XML based Remote Procedure Call (RPC) communication from J2ME. The included packages are subset of the JSR 101 specification, JAX-RPC.

6.3 Microsoft .NET

Microsoft .NET [NET] is Microsoft's answer to the next generation of distributed networking, a "software platform" where software is delivered as services. It is a language-neutral environment for writing programs that can easily and securely interoperate. Programs are not targeted a particular hardware and operating system combination, but will run wherever .NET is implemented. The intention is that user should be able to access their information across all devices.

Microsoft .NET is a set of Microsoft software technologies for connecting the world of information, people, systems, and devices. It enables software integration through the use of XML Web Services, as well as to other, larger applications, via the Internet. Microsoft .NET spans clients, servers, and services, and consists of [NETfto]:

- .NET Framework: A programming model that enables developers to build Extensible Markup Language (XML) Web services and applications.
- A set of XML Web services, for example Microsoft .NET My Services.
- A set of servers, including Windows 2000, SQL Server, and BizTalk Server, that integrates, runs, operates, and manages XML Web services and applications.
- Client software, such as Windows XP and Windows CE.
- Tools, such as Visual Studio .NET, to develop XML Web services and Windows-based and Web applications.

.NET Framework is the programming model, or infrastructure, for the .NET platform. Microsoft explains the .NET Framework as an "environment for building, deploying and running Web services and other applications." [NETfaq]. Basically, the .NET Framework is just a single platform that anybody can develop for, using a system similar to that used in Java. The difference is that any language could be used, also within a specific program. The current available languages are Managed C++, C#, Visual Basic and JScript, and more are about to come.

The framework consists mainly of two parts:

- The Common Language Runtime (CLR)
- The .NET Framework class library

In addition there is a third part; the applications that can range from traditional command-line or graph-

ical user interface (GUI) to applications based on ASP.NET, such as Web Forms and XML Web Services. ASP.NET is a programming framework for building web-based applications. The .NET Framework provides a fully managed, protected, and feature-rich application execution environment, simplified development and deployment, and seamless integration with a wide variety of languages. An overview of the .NET Framework is shown in Figure 6-4.

The *Common Language Runtime (CLR)* is the foundation of the .NET Framework. It is responsible for runtime services such as language integration, security enforcement, and memory, process, and thread management. It also manages code at execution time, and enforces code robustness by implementing a strict type- and code-verification infrastructure called the Common Type System (CTS). In addition, it has a role at development time with features such as lifetime management, strong type naming, cross-language exception handling, and dynamic binding. This reduces the amount of code a developer must write to turn business logic into a reusable component ([NETfto], [NETfaq]). The Common Language Runtime is displayed as the lowest layer in Figure 6-4.

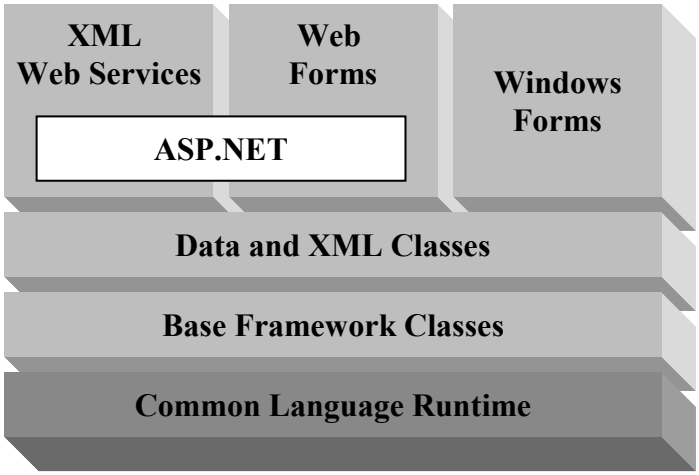


Figure 6-4 Microsoft .NET Framework

The *.NET Framework class library* is a comprehensive, object-oriented collection of reusable types that you can use to develop applications, for example by usage of ASP.NET. The library consists of five class libraries, as visualized in the upper three layers of Figure 6-4:

- Base Framework Classes
- Data Classes
- XML Classes
- XML Web Services
- Web Forms
- Windows Forms

The *Base Framework classes* provide standard functionality such as input/output, string manipulation, security management, network communication, thread management, text management, and other similar functions. *Data classes* support persistent data management and include SQL classes for manipulating persistent data stores through a standard SQL interface, while the *XML classes* enable XML data manipulation and XML searching and translation. The three last libraries are dependent upon the Data and XML classes. *XML Web Service classes* support the development of lightweight distributed components, which will work even in the face of firewalls and network address translation (NAT) software. *Web Forms classes* include classes that enable rapid development of Web graphical user interface (GUI) applications, while *Windows Forms classes* support a set of classes that allow you to develop Windows-based GUI applications

that provide a common, consistent development interface across all languages supported by the .NET Framework. ([NETov], [NETfto]).

Currently, .NET is only available on Windows, but Microsoft claims that is will be available on Linux some time in the future. In a worst-case scenario .NET will never migrate beyond the Windows world; otherwise it might become an alternative development platform to Java.

6.3.1 .NET for Java developers

Microsoft .NET is also available for Java-language developers through Microsoft Visual J# [NETJ#], Microsoft's Java development tool for the Windows platform. J# is Microsoft's implementation of the Java programming language. An existing Java application can also be moved to the Microsoft .NET Framework with use of Microsoft Java Language Conversion Assistant [JLCA]. The tool will automatically convert existing Java-language source code into C#. Or at least this is what Microsoft claim. [NET-Java].

6.3.2 .NET Compact Framework

The .NET Compact Framework is a subset of the .NET Framework that is designed to run on smart devices, providing support for managed code and XML Web Services. It enables the execution of secure, downloadable applications on devices such as personal digital assistants (PDAs), mobile phones, and set-top boxes. In this way it is a key part in Microsoft's goal to provide customers with great experiences at any time, any place and on any device. ([NETcomp], [telecom01])

.NET Compact Framework is designed to run on multiple hardware platforms and operating systems, but will first be available on devices running the Microsoft Windows CE operating system. This includes Pocket PC and Pocket PC 2002 devices, Microsoft Smartphone, and devices running Windows CE .NET. No other supported platforms are announced at the moment. Languages that will be supported are Microsoft Visual C# and Microsoft Visual Basic. Other .NET languages will run as long as they conform to the subset of the Common Language Runtime (CLR) that the .NET Compact Framework supports. [NETcfaq]

J2ME and the .NET Compact Framework is in other words competitors in the world of small devices. But until .NET Compact Framework supports both Java and a multitude of platforms, J2ME is more attractive to Java-developers.

6.4 Summary

This chapter presents the AMIGOS project and two upcoming technologies that relate to Jini network technology and JXTA.

AMIGOS (Advanced Multimedia In Group Organized Services) is a service that provides users with the possibility to establish and participate in Meeting Places, i.e. virtual rooms where interactions optionally take place between the participants. A Meeting Place might be limited to geographical areas, and interactin includes exchange of information and media streams like text, audio and graphics.

Web Services is a service architecture where applications are able to communicate regardless of implementation language, platform they were developed for, and the object models and internal protocol they use. The Web Service Platform consists of four parts that forms the common ground. Simple Objecct Access Protocol (SOAP) is based on Extensible Markup Language (XML), and provides a way to invoke services across the Internet. Web Service Definition Language (WSDL) describes operational information, i.e. where the service is located, what the service does, and in what manner a requester can make a connection to the service. The UDDI database stores and distributes references about the offered Web Services, defines a way to describe a company's capabilities, and a way for companies to queri each other's

capabilities. The platform is independent on the transport protocol used to transfere the exchanged data. The primary Web Service model, the Service Oriented Architecture (SOA), assignes different roles to the involved parties in a Web Service like Requester, Provider and Broker. An API for J2ME is under development to provide access to Web Service for J2ME enabled devices.

Microsoft .NET is Microsoft's answere to the next generation of distributed networking, and is built up around many components like the .NET Framework, XML Web Services, Servers, Clients and Tools. The .NET Framework is the programming model, or infrastructure, for the .NET platform, and consists of two parts: the Common Language Runtime (CLR) and the .NET Framework class library. CLR is responsible for handling the runtime services like language integration, security and resource management. The .NET Framework class library is a collection of reusable types that can be used to develop applications. It consists of several libraries that provides standard functionality, persistent data management (SQL), XML data manipulation, and libraries for application development. Microsoft .NET have an implementation for PDAs and mobile phones called .NET Compact Framework that is a subset of the .NET Framework.

Chapter 7

The Prototype

"Start by doing what's necessary,
then do what's possible,
and suddenly you are doing the impossible."
St.Francis of Assisi

A prototype application was developed as part of this master thesis. The goal was to illustrate some of the features of JXTA for J2ME in use with MIDP, and how data services can be used on next generation of mobile terminals. As a result, the prototype is based upon instant messaging.

Instant messaging is a well-known way to communicate with other people on the Internet. Because of the many ways to realize such communication, many is trying to standardize how instant messaging is implemented so as to ensure compatibility between different implementation. The standardization effort is presented in Appendix F.

This chapter will go through all the phases of the development process, from analysis and requirements specification to design, implementation and test. The modelling language used is the Unified Modelling Language (UML). Analysis is done through service specification including user needs and scenarios, and is the basis for the requirement specification. Use case diagrams accompany the requirements specification by visualize the functional requirements and their dependencies. Class diagrams, sequence diagrams and collaboration diagrams are used in the design phase. Next it will explain some of the implementation solutions, and suggest alternative solutions for some cases, before turning to the test specification and related results.

The prototype is an Intelligent Traffic Service (ITS) system, as suggested by professor Do Van Thanh at Department of Telematics (ITEM) at NTNU. It was developed using the Java Wireless Toolkit (JWT) [J2MEwtk] from Sun, and tested on the included emulators.

7.1 System overview

People travelling with cars (or other vehicles) may be interested in getting information about the area they are travelling in. This might be information about a new road, detours because of traffic accidents or roadwork, or simply to get explanation about the way if a map is out of reach.

Some solutions already exist, like radio programs dedicated to travelling people, informing about things to pay attention to when driving. These programs are most widespread and useful in big cities, and the information might be useless if you are not in the specific area where the information is relevant. The same is the case when a driver is on a long distance trip and listens to a radio program passing on travelling information for drivers in the whole country. Other drawbacks are that these radio programs are generally only broadcasted during holidays, or in rush hours.

The Intelligent Traffic Service (ITS) system aims to solve these problems. A client running the service should be available for downloading to the users mobile device over the air, and installed with little or no interaction from the user. The terminal must support J2ME CLDC/MIDP. Figure 7-1 illustrates a situation where this service could be useful for other travellers.

Figure 7-1 A user informing other users about queue in an area

7.1.1 Service Specification

The idea of this service is to let travelling people communicate with other in the same area or with the same destination target. For instant if a user is driving a car in a specific area and wants to get information about the traffic there, the user can register to a group representing this area, and receive information or ask questions to everyone else registered in that group, or only to one specific user in the area. The difference between unicasting and multicasting the message will be described later. Another example is the case of a user travelling from Oslo to Trondheim that may need to exchange information with other travellers on the same journey. Figure 7-2 shows a scenario were a user is looking for a gas station in a specific area.

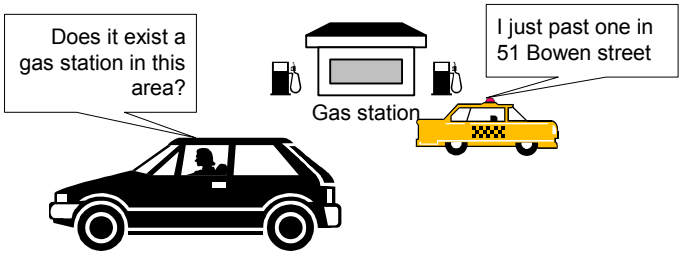


Figure 7-2 Example of usage of the service

The call outs are actually text written on a mobile terminal, for instance by a passenger. In principle the application could be controlled both by speech and physical interaction from the user, since a user driving will not be able to write messages by hand while driving. Speech recognition is on its way, but there is still a long way to go, and as a result this version of the prototype will only realize the possibility to send text.

Group belonging for a user could be decided based on location information supplied by for example Global Positioning System (GPS). Such a solution allows the device to find information about the location without the need for a user to supply the device with this information. Unfortunately this is a scenario for the future, but this prototype will just as well be a locationbased service. In contrast to GPS or similar technology, called second generation of locationbased equipment, this service will be of first generation. This means that the user is responsible for typing in the location to join a suitable group.

As the service is location based, groups basically represents locations, but could in practice represent other areas of interest. In this way the service become a generic service that can be used for much more than the initial thought.

7.1.2 Realization

To use this service the user must log into the service with a username and then decide whether to join a public or private room.

In a public room the messages will be sent to every member of the group, while in the private room there will be an exchange of messages between two persons in the same group. The user then selects one of the users in a contact list, and sends an instant message to this user. The two last figures have illustrated examples of messages, which could be sent in a public room. Figure 7-3 shows a scenario where it could be favourable to be in touch with other persons travelling the same route, and where instant messaging would be the preferable chatting mode.

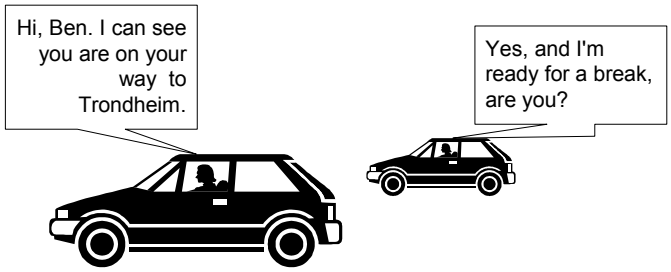


Figure 7-3 Example of usage when chat mode is private

To be able to communicate with other in the same area, the users have to join a group representing this specific area or distance. Either the user accesses a list of possible groups, which exists and picks one of them, or the user searches for a group by name. If that group does not exist, a new group can be created.

7.2 Requirements Specification

In the requirements specification phase of the system development all the requirements of the prototype are identified. Functional requirements specify what the system should be able to do, that is the functionality, without focusing on the physical implementation. Non-functional requirements are the implementation requirements and focuses on factors like performance, robustness, reliability, etc. UML use-case diagrams will describe how a system will look to potential users, but is most useful to communicate among the developers to obtain a common understanding of the system. A use-case diagram is a collection of scenarios initiated by an entity called an actor (a person, a server, etc.), and each functional requirement is described by a use-case diagram.

The actors identified in the ITSSystem's environment are:

- **The initiator**: This is the main user in the environment, and is the user holding the mobile device and using the client in the ITSSystem.
- **The terminator**: The initiator can connect directly to another user using ITSSystem. This other user is called the terminator.
- **The peer group**: This is an abstraction of all possible users in the ITSSystem environment. When "peer group" is used in the use case diagrams, it signalizes that all users are involved to fulfil a requirement.

7.2.1 Functional Requirements

This section described the requirements that are made to the functionality of the prototype application. The functional requirements will be described by text, and illustrated in UML use case diagrams. Each functional requirement is assigned an identity in the format F - <number>, where <number> is a sequential number.

F - 1 Login
User shall be authenticated prior to service usage. The user is required to log on to the service at start up. The login includes an authentication of the user based on username.

F - 2 Modus of communication
The user can choose whether to be in the public mode of communication, where messages are received and sent to the whole group, or in private mode, where the messages are exchanged between two users participating in the same group. The user select between the two modus before joining a group.

F - 3 Change mode of communication
When the user is connected to a group, the user shall be able to change the communication mode from public to private, or the other way around.

Figure 7-4 shows the use case diagram for F-1, F-2, and F-3.

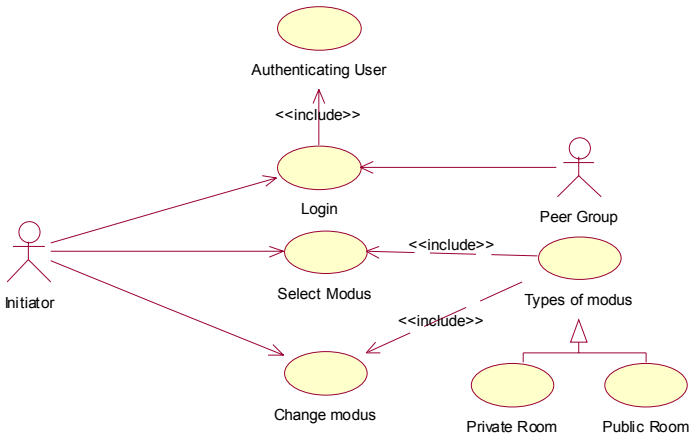


Figure 7-4 Use case diagram for login and chat mode

F - 4 Send messages
The user shall be able to send messages to a group (public mode), or to a specific person in the contact list (private mode).

F - 5 Receive messages
The user shall be able to receive messages from members of the current group when in public mode, and from members in the contact list when in private mode.

F-4 and F-5 is illustrated in Figure 7-5. When the user is in private mode, the sending and receiving of messages involve a second use (the terminator), while in the public mode, the message is sent to and received from the whole group (the peer group). A message to be sent can both be a new message, or a reply to an incoming message, when in private mode

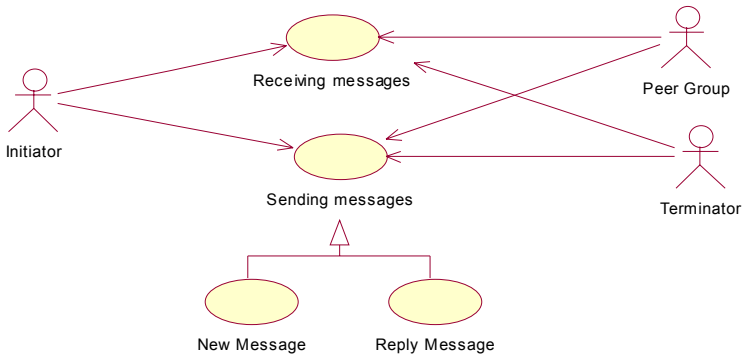


Figure 7-5 Use case for sending and receiving of messages

F - 6 View group list
The user shall be able to view a list of groups. The list consists of the groups of interest for the user.

F - 7 Join group
The users shall be able to join a group. To join a group, the user selects one of the groups in the group list.

F - 8 Change group
The users shall be able to switch from one group to another. This includes disconnecting from the previous group and joining a new group.

F-6, F-7 and F-8 is shown in Figure 7-6. In addition a requirement for editing the group list is shown. This is covered in the next requirements.

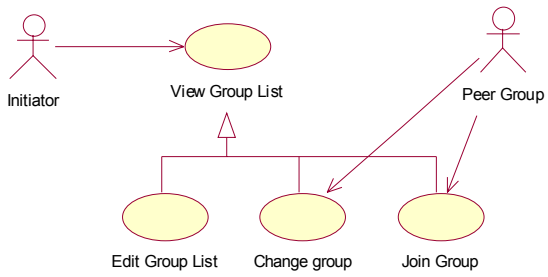


Figure 7-6 Use case diagram for viewing the group list

F - 9 Edit group list
The user shall be able to edit the group list, by deleting groups and adding new groups. This means editing the local list located in the user's database.

Figure 7-7 shows F-9 and also the next requirement, F-10 Add group, which is a sub requirement of F-9.

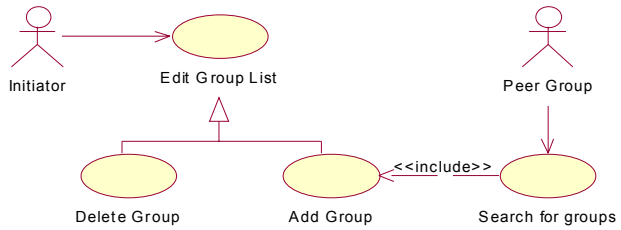


Figure 7-7 Use case diagram for group list viewing and editing.

F - 10 Add group
Users shall be able to add new groups. This includes first of all searching for the group in the network, and if the group does not exist create the group.

F-10 is illustrated in Figure 7-8. When the user wants to add a group, a search among the existing groups is initiated. If no group are found, a new group is created.

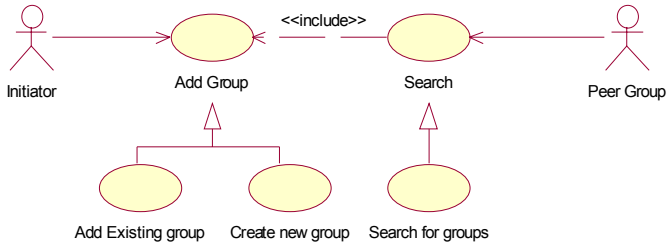


Figure 7-8 Use case diagram for adding a group

F - 11 View contact list
The user shall be able to see a list over personal contacts. This list will only contain the contacts that are online and participating in the same group as the user.

Figure 7-9 illustrates F-11 and shows that a user may edit the lists, as stated in the next requirement.

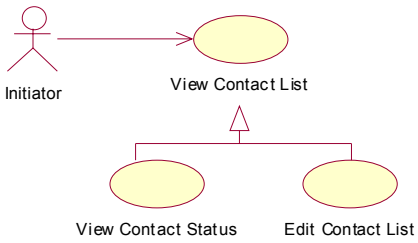


Figure 7-9 Use case diagram for viewing the contact list

F - 12 Edit contact list
The user shall be able to edit the contact list, either by deleting contacts or adding new one by searching for contacts in the group.

F-12 is illustrated in Figure 7-10 and is identical to the use case diagram for requirement F-9.

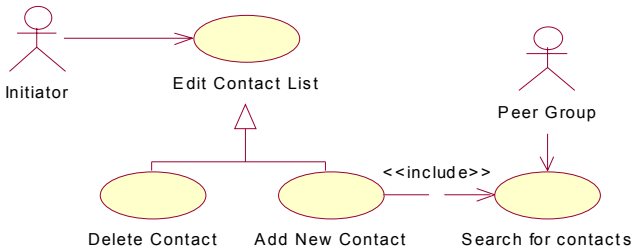


Figure 7-10 Use case diagram for editing the contact list

A use case diagram showing the entire system can be found in Appendix C.

7.2.2 Non-functional Requirements

The non-functional requirements for the ITSSystem includes both general requirements for the development, and some performance attributes. Performance attributes are included to be able to validate the technology, but is not of vital importance for this prototype. Later versions should seek to fulfil these requirements. Each non-functional requirement is assigned an identity in the format NF - <number>, where <number> is a sequential number.

NF - 1 Network technology independence
The network technology shall be hidden from the user, and be independent of the application so that any network technology can be used. Network technologies can be client-server protocols and peer-to-peer technology. One should easily be able to change a JXTA for J2ME implementation with a HTTP and central server implementation.

NF - 2 Run in SDK and JWT
The prototype should run both in the Software Development Kit (SDK) and the Java Wireless Toolkit from Sun.

NF - 3 Run on a J2ME compliant device
The prototype should be able to run on a cellular phone or PDA supporting J2ME, for instance the Motorola Accompli 008.

NF - 4 Usability
The user should be able to use the service application without training, and should get to know it within 10 min.

NF - 5 Security
Information that is exchanged between two parties in private room in the ITSSystem should not be available for other parties than the parties involved in the session.

NF - 6 Availability
The user should be able to connect to the network and register with the service in 90% of the cases.

NF - 7 Reliability
The service should run without problems for 12 hours without the user needing to start the application over again and login to the service. If disconnected, the user should be able to continue his session when logging on within 5 minutes.

NF - 8 Cost
The user should be able to control the cost of using the service, for instant by choosing how often to poll for incoming messages.

NF - 9 Response time
The user should receive response from the relay within the set poll interval (which can be set by the user).

7.2.3 Requirements and prototype versions

When implementing the system it is important to break the system up into smaller parts. In that way not everything need to be implemented at the same time, and testing of a particular requirement will be easier. This is the reason for making versions of the system, where the first version will only satisfy some of the requirements, the next version will implement additional of the requirements and so on. In the last version, all the requirements will be satisfied. The goal is that the prototype developed in this project should be of version 1.0.

Version 1.0
- F - 1 Login
- F - 2 Modus of communication
- F - 4 Send messages
- F - 5 Receive messages
- F - 6 View group list
- F - 7 Join group
- F - 8 Change group
- F - 9 Edit group list
- F - 10 Add group

By version 1.0 the prototype should support enough functionality to work as a stand-alone service and be useful for the users.

The first rollout will include features for the user to login with a username. No password is required. Communication can only go through public room in this version, and the user will be presented for a list of groups and should be able to join one of these groups. The group list is stored in a database, where it can be retrieved and stored at start-up and closure respectively. In addition the user should be able to add an existing group to the group list, and change between the different groups in his list. Groups can also be deleted from a users group list.

The user should be able to send messages to the group, and receive message from the group he is currently participating in. The total number of messages visible at all time might be set by the user in setup, as well as the polling frequency.

Version 1.1
- F - 1 Login
- F - 2 Modus of communication
- F - 3 Change mode of communication
- F - 11 View contact list

Version 1.1 introduces the second mode of communication, the private room. The user can choose between public and private mode in setup, and at any time change mode. Public mode is the default mode. When entering the private room, the user is presented for a list of contacts. A user in a private room shall also join a group, and can only communicate with the contacts participating in the same group (the communication will be supported in version 1.2).

Version 1.2

- F - 4 Send messages
- F - 5 Receive messages

A user should be able to send and receive messages from users in the contact list while in private mode.

Version 2.0

- F - 8 Change group
- F - 11 View contact list
- F - 12 Edit contact list

Version 2.0 supports the main functionality of private room in addition to the functionality of public room supported in 1.0. The database has to be extended to support saving of the contact list, and to retrieve the contact list at start-up. A new contact might be added to the contact list and saved to the database, and a user must be able to delete contacts from the list.

7.2.4 Summary of requirements

Table 7-1 summarizes the requirements for the ITSSystem.

Number	Name
Functional Requirements	
F - 1	Login
F - 2	Modus of communication
F - 3	Change mode of communication
F - 4	Send messages
F - 5	Receive messages
F - 6	View group list
F - 7	Join group
F - 8	Change group
F - 9	Edit group list
F - 10	Add group
F - 11	View contact list
F - 12	Edit contact list
Non-functional requirements	
NF - 1	Network technology independence
NF - 2	Run in SDK and JWT
NF - 3	Run on a J2ME compliant device
NF - 4	Usability
NF - 5	Security
NF - 6	Availability
NF - 7	Reliability
NF - 8	Cost
NF - 9	Response time

Table 7-1 Prototype requirements

7.3 Design

This section will go through the design phase, and is illustrated by class-, state-, sequence-, and collaboration diagrams. A class diagram shows how the classes of a system relate to one another, while the state diagram focuses on the state changes in just one object. While the sequence diagrams shows how object communicates with each other over time, collaboration-diagrams shows how objects interact accordance to space.

7.3.1 The Architecture

The architecture is dependent on the network technology used. This prototype will use the JXTA network technology presented in chapter 5, and the JXTA for J2ME (JXME) package that is designed for J2ME enabled devices.

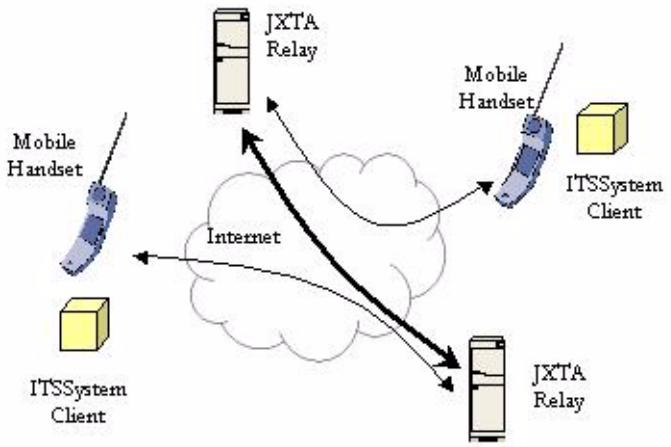


Figure 7-11 The architecture of the ITSSystem

Mobile devices, represented as cell phones in Figure 7-11, are dependent upon a relay/proxy server on the Internet to connect to the JXTA network. The proxy servers are responsible for forwarding messages to the appropriate ITSSystem client by sending it to the proxy server where that particular client is registered.

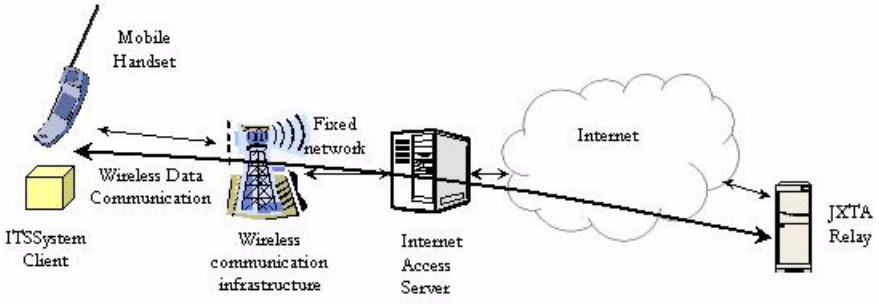


Figure 7-12 A cell phone connect to the proxy

Network technologies used for transport will influence the way a mobile terminal will connect to the relay. Figure 7-12 shows a scenario for a cell phone using GSM or GPRS. The client will then have to use to connect via a wireless link to a terminal access server before connecting to a proxy server. This is the

most likely scenario for drivers, but the hope would be that the phone itself could decide the most appropriate transport technology to use at any given time and place, and change this when appropriate.

7.3.2 Class Diagrams

The service consists of three main classes: the NetworkClient, the DatabaseClient and the ITSClient. NetworkClient is responsible for the network connection, while the DatabaseClient is responsible for interacting with the database on the mobile device. ITSClient is the main class and takes care of the interaction with the user through a graphical user interface (GUI), and uses NetworkClient and DatabaseClient to obtain necessary information.

A class diagram giving the overview of the system, hiding attributes and operations (methods), is illustrated in Figure 7-13. The responsibility for implementing the different parts is also shown as notes in this diagram.

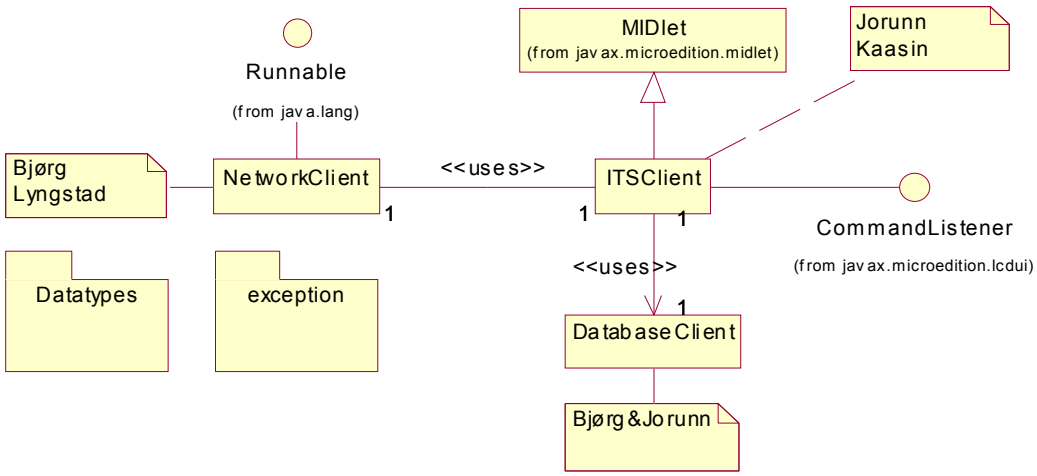


Figure 7-13 Overview of the class diagram for the ITSSystem

Every interaction the network is done through the NetworkClient. This class will be implemented using JXTA for J2ME, and will then be able to interact with any device on the JXTA network. NetworkClient takes care of the basic tasks associated with peer group membership, like group- and user discovery, and the joining and creation of groups.

Since the only supported protocol in MIPD for communication over network is HTTP, the NetworkClient uses a JXTA proxy. At specified intervals it asks the proxy if messages has arrived for the user. NetworkClient implements java.lang.Runnable and creates a thread for the task of polling the relay. The public attributes and methods of NetworkClient are shown in Figure 7-14.

The non-functional requirement NF-1 states that the system should be independent of the network. The realisation of the NetworkClient fulfils the requirement, since it hides the interaction with the network for the user, and all exchange of data inside the system is system-specific, not network specific. Communication and exchange of information and data is done using an ITSSystem specific datatype called ITSMessage, included in the Datatypes package. The class diagram for ITSMessage is shown in Figure 7-15.

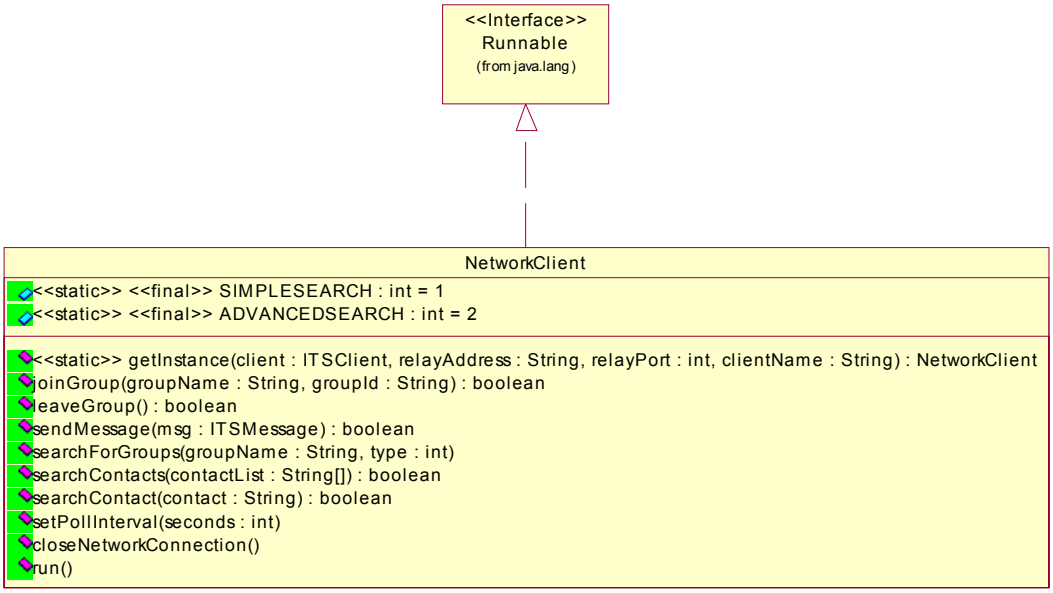


Figure 7-14 Detailed view of NetworkClient

The ITSClient extends the MIDlet class of J2ME MIDP, and since MIDlet is the basic unit of execution in MIDP, ITSClient is responsible for the application entry and leaving of states during the MIDlets whole life cycle. The different states are pause, active and destroyed. To react to the different command actions received from the user, ITSClient implements the CommandListener interface.

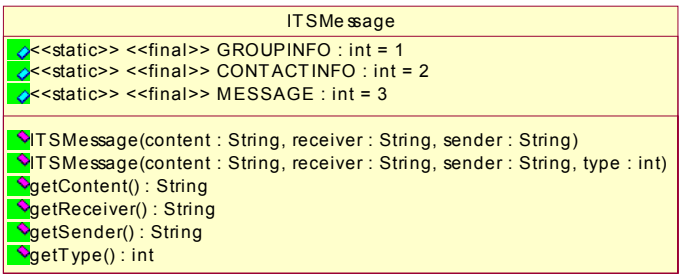


Figure 7-15 The datatype ITSMessage

The ITSClient will implement the graphical user interface and includes all the methods used to create the screens with the appropriate text, button, lists and textboxes. ITSClient will use the DatabaseClient to retrieve and store information to the database, and use the NetworkClient to communicate with the JXTA network. The only public method, beside the constructor and inherited methods, is receiveMessage (ITSMessage). This method is called by the NetworkClient when an incoming message has arrived for the user. The class diagram for the ITSClient and the DatabaseClient is illustrated in Figure 7-16.

The DatabaseClient uses the Record Management System (`javax.microedition.rms` package) of MIDP to achieve a mechanism for MIDlets to persistently store data and later retrieve it. `RecordStore` is the class representing the collection of records that contributes the database. The database will consist of contacts, contact ids, groups and group ids.

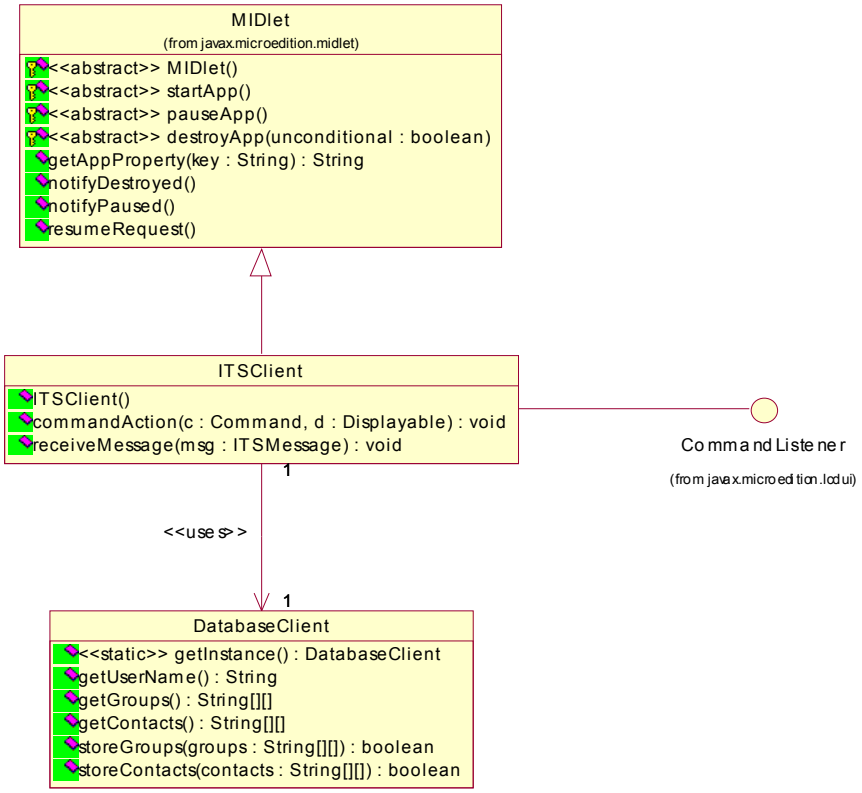


Figure 7-16 Detailed view of the ITSClient and the DatabaseClient

To further make the ITSClient independent of the network and storage and retrieval of data, some ITSSystem specific exception classes are introduced. The hierarchy is shown in Figure 7-17.

The detailed class diagram can be viewed in Appendix C.

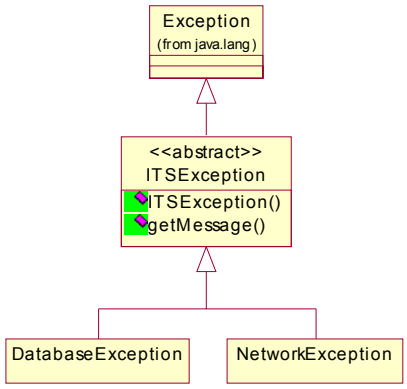


Figure 7-17 The exceptions in the ITSSystem

7.3.3 State Diagram for the ITSClient

Figure 7-18 shows the state diagram for the ITSClient. Regular UML syntax is used, so square brackets represent conditions and ovals represent states.

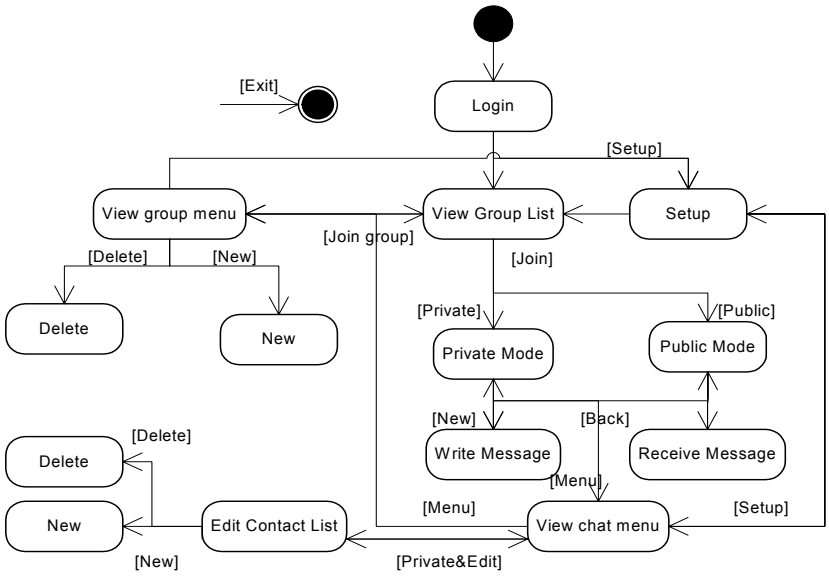


Figure 7-18 State diagram for the ITSClient

The state diagram can to a certain degree be translated directly to the screen flow, illustrated later in this section, but the states concerning communication in private mode is left out.

From the diagram is seems that it is only possible to receive messages (that is messages from other users) when the user is either in the states of private and public mode. This is not totally correct, since the user can receive messages when viewing the chatting menu or writing a new message as well, but the message will only be visible when returning to the communication screen, either in public or private mode.

When the user is in the state calls "View group menu", no groups are joined, and the user get the possibility to edit the group list, to join a group or to view and edit the setup.

In "View chat menu" the user sees a menu, much the same as the group menu, but with some other selection options. The "View chat menu" state gives the user a possibility of selecting among several alternatives:

- Change group
- Add group
- Search contact (if in private mode)
- Delete contact (if in private mode)
- Setup

In this way the user can always add new or delete existing groups or contacts, and edit the setup easily.

The diagram does not consider possible error situations. "Exit" will always end the program.

7.3.4 Sequence- and collaboration diagrams

Sequence diagrams and collaboration diagrams illustrated the interaction between objects in a system, in time and space respectively. Since these diagrams show the exact same information, this section will il-

lustrate some of the communication that occurs between object with sequence diagrams and others with collaboration diagrams. All diagrams will be found in Appendix C.

Figure 7-19 shows the login sequence, involving the ITSClient and the DatabaseClient. The client must first create an object of the DatabaseClient and then retrieve the user name from the database, used to authenticate the user. Where the username shall be set and where to keep it has some alternative solutions that will be covered in the implementation notes.

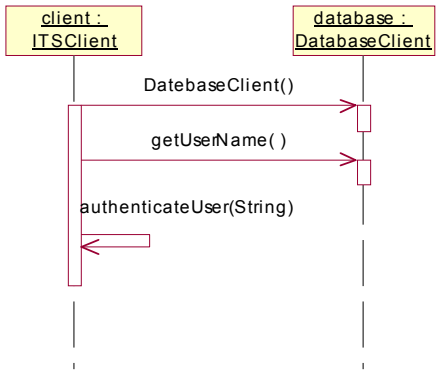


Figure 7-19 Sequence diagram for login and authentication

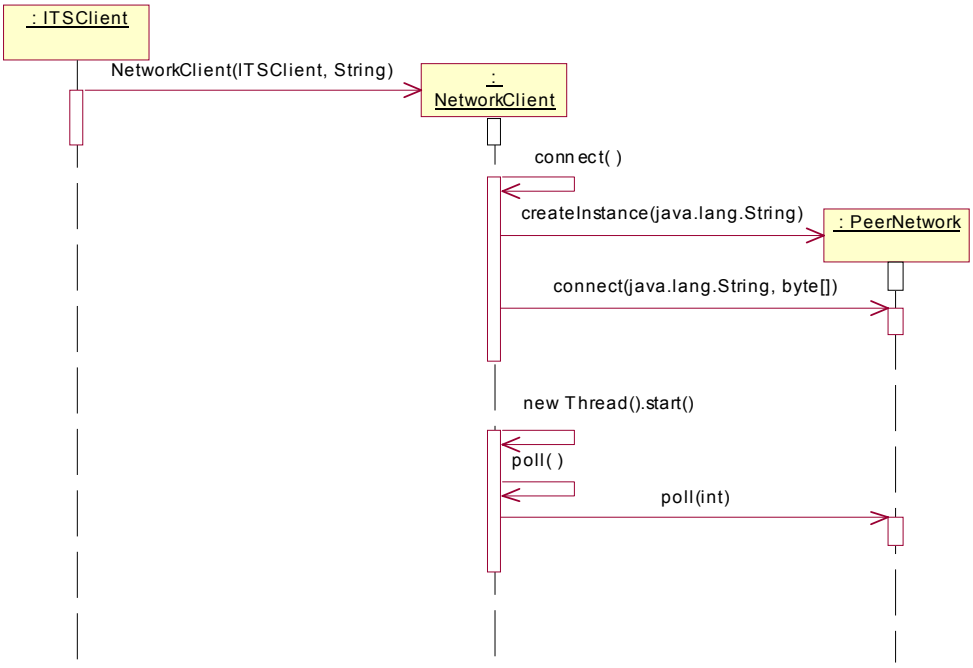


Figure 7-20 Sequence diagram showing how to connect to the network

For the client to connect to the network it has to create an object of the `NetworkClient`. This is shown in Figure 7-20. NetworkClient will create an instance of `PeerNetwork`, one of the classes of the JXTA for J2ME package, and then make this connect to the network by calling `connect()`. To be able to receive messages, the NetworkClient must poll the relay/proxy, and this is done using a thread that is initialized and started when the NetworkClient is initialized.

The joining of a group is illustrated in Figure 7-22. The numbers in the collaboration diagram indicates the messages' order in the sequence, and conditions are represented with square brackets. When the user decides to join a specific group, the NetworkClient will start listening to the pipe where messages from the group are sent. Pipes and other features of JXTA are presented in Chapter 5. If the user is already member of a group, the NetworkClient will call the `leaveGroup()`-method before starting to listen to the new pipe representing the new group.

Messages are retrieved by polling the proxy. If the proxy has received some messages to the client, this message is forwarded to the ITSClient as an ITSMessage.

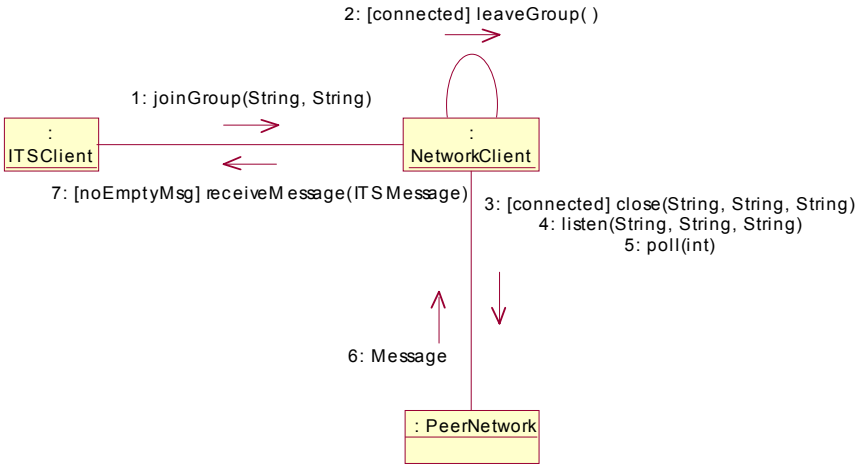


Figure 7-21 Collaboration diagram showing the sequence of joining a group

Figure 7-22 illustrates the viewing and editing of a group list. The first thing happening is that the client gets the groups from the database. The second operation (2) is executed if the user chooses to view the group menu. Adding new groups involves call to the `searchForGroups()`-method of NetworkClient, which asynchronously returns an ITSMessage with groupName and groupId
.

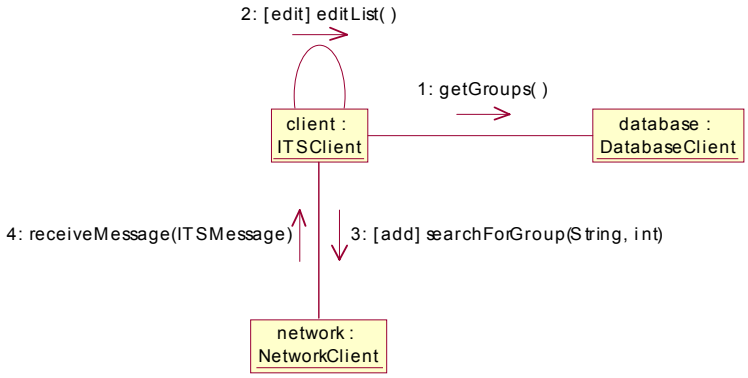


Figure 7-22 Collaboration diagram for viewing and editing a group list

Adding a group to a contact list may involve a number of steps. When the ITSClient calls network.search-ForGroups (groupName, typeOfSearch), the NetworkClient must first check whether the group already exists or not. The sequence of adding an existing group is shown in Figure 7-23.

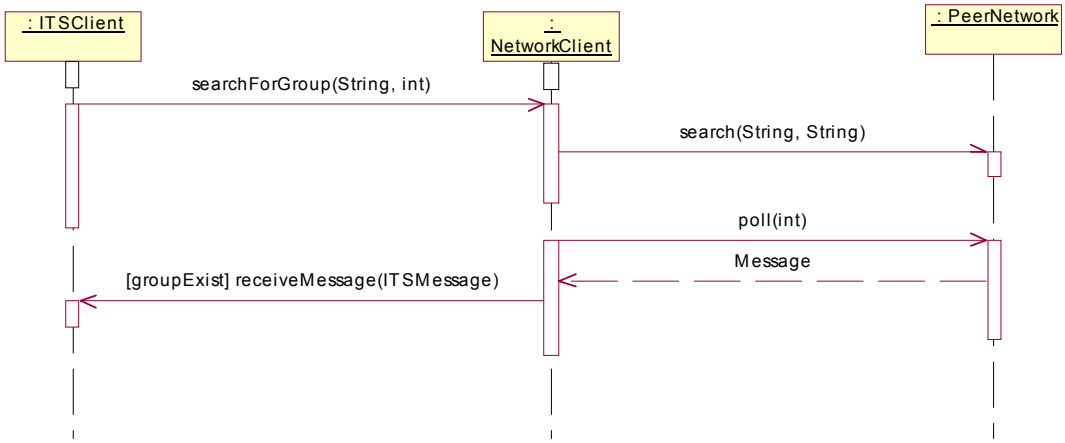


Figure 7-23 Sequence diagram for adding an existing group

When a user wants to add a group to the group list, he will supply a group name and indicate whether the group must have the exact same name or similar name of the supplied group name. The NetworkClient will then initiate a search to find the id to the group or similar groups. When searching for similar groups, the resulting groups are added to a result list that the user can examine at a later point. From this list the user can choose the group to be added to the group list, and then later choose to join the group.

If the network client receives a message containing information about a suitable group, this information is passed to the ITSClient. If the search was for an exact match, the group is added to the group list. If no message about an appropriate group is received within a predefined time interval, the group is considered non-existing, and a new group will be created if the user chooses to join the group at a later time. This is illustrated in Figure 7-24.

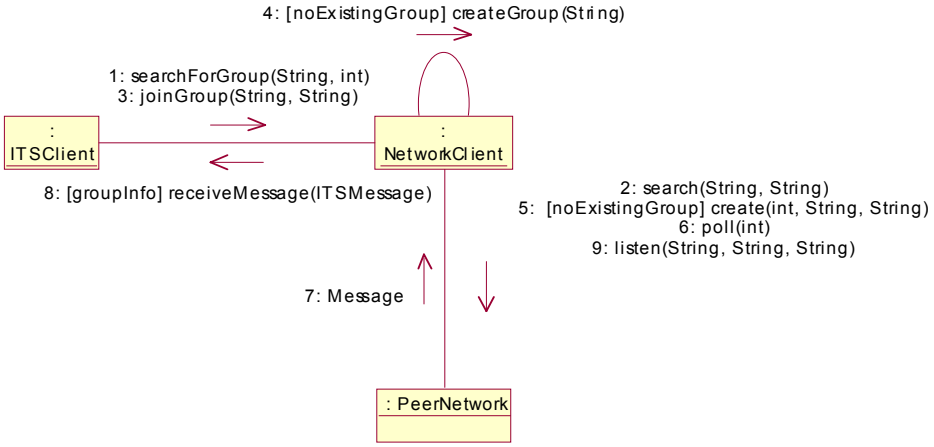


Figure 7-24 Collaboration diagram for adding a non-existing group

If a user chooses to participate in a private room instead of a public, the list of contacts is selected from the database. Since only the contacts which are reachable in the same group as the user will be presented for the user, the client need to check which contacts are participating in that group at this moment. This is done by calling the method in NetworkClient called `searchContacts()`. For each contact that is online, the client will receive a message containing information about the contact, and the contact is add-

ed to the users visible list. This is illustrated in Figure 7-25.

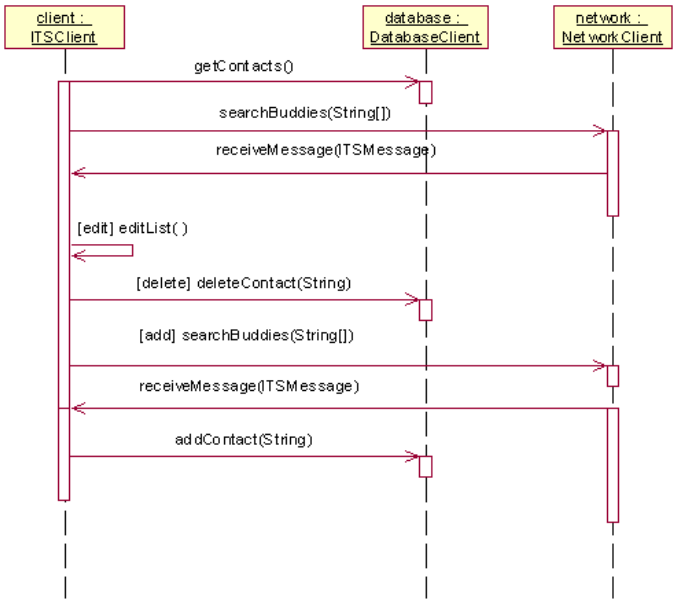


Figure 7-25 Sequence diagram for editing a contact list

If the user wants to delete one of its contacts, this is done by removing the user from the internal contact list of ITSClient. The changes will be reflected in the database after storing the list to the database upon closing the application. On the other hand, if the user wants to add a new contact, the NetworkClient need to search for the contacts in the JXTA network, which in turn will return information about the contact to the ITSClient. The contact is added to the internal contact list, and stored in the database upon closing the application, as shown in Figure 7-27.

When the user is either participating in a public or private room, it can start sending and receiving messages. This is shown in Figure 7-26.

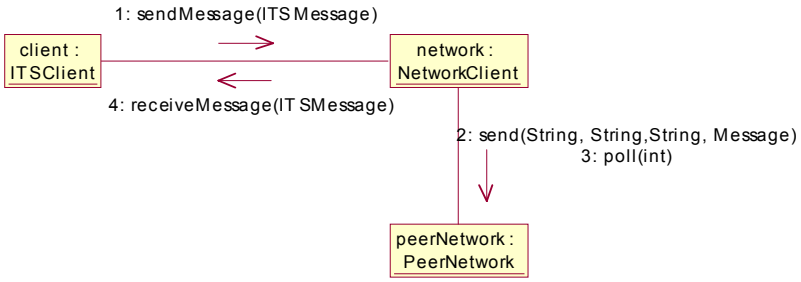


Figure 7-26 Collaboration diagram for sending and receiving messages

When the user wants to end the running of the application, the client will tear down the connection to the network through the NetworkClient. The NetworkClient will unsubscribe from the pipe it has been listening to. This is illustrated in Figure 7-27.

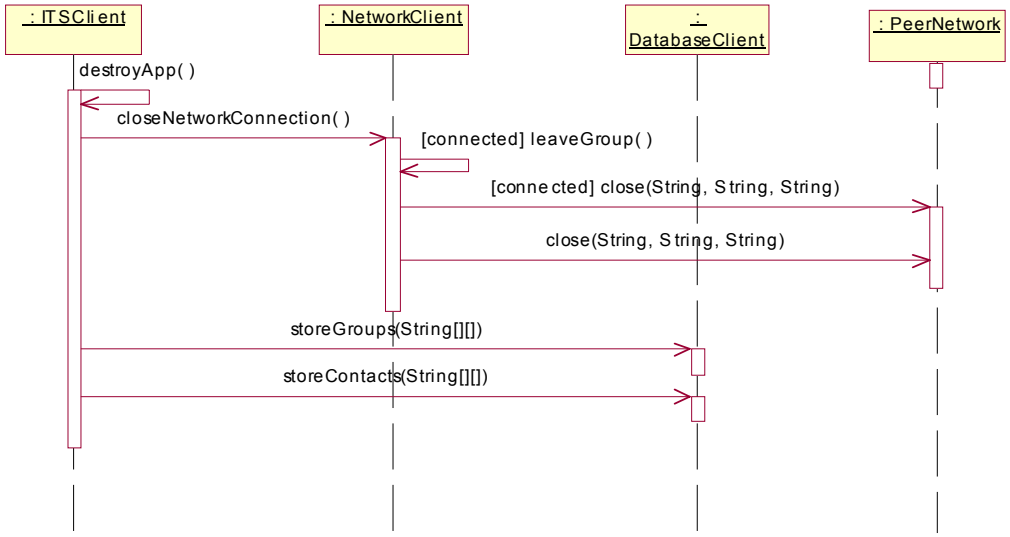


Figure 7-27 Sequence diagram for exit the application

All the remaining UML diagrams can be found in appendix C.

7.4 Implementation

The prototype was implemented using J2ME Wireless Toolkit [J2MEwtk] and the supplied emulator in this toolkit. The code was written in emacs, and includes comments to the implementation.

The J2ME Wireless Toolkit [J2MEwtk] offers a lightweight way of building and testing a MIDP project. It comes complete with the CLDC and MIDP libraries, utilities, Javadoc documentation and example Midlets. The toolkit runs on any Java 1.3 enabled platform and is free of charge. It also allows developers to test their applications on different emulated target devices. The wireless toolkit is presented in more detail in Appendix E.

7.4.1 Screen flow

To get a better understanding of the system and the implementation issues discussed next, this subsection will presents the user interface of the system, and the flow between the different screens in the user interface. First, the focus is on the implemented version of the ITSSystem, and then we turn to the proposed user interface for the rest of the system.

Figure 7-28 shows the flow between the different screens in ITSSystem version 1.0. The screens displayed are those available for the user, and should be viewed to possible customer to check the usability of the program.

Note that one error-screen is shown as an example. In this case the user is trying to add a group without specifying the name of the groups, which is a non-valid situation. This is brought to the users attention by specifying the type of error. The error-screen can appear when other error situations occur, but with different messages according to the situation.

The screen with the title “Join group” is shown twice in the diagram because it has to appearances, first is when no groups exist, and next is when they are. When no groups exist, the OK button is unavailable as the possibility of joining a group is not present. The same screen is also used then users chooses to

delete groups, and here as well is the OK-button removed when no groups exists. When there exists group in the list the possibility to join one of the groups is made available for the user.

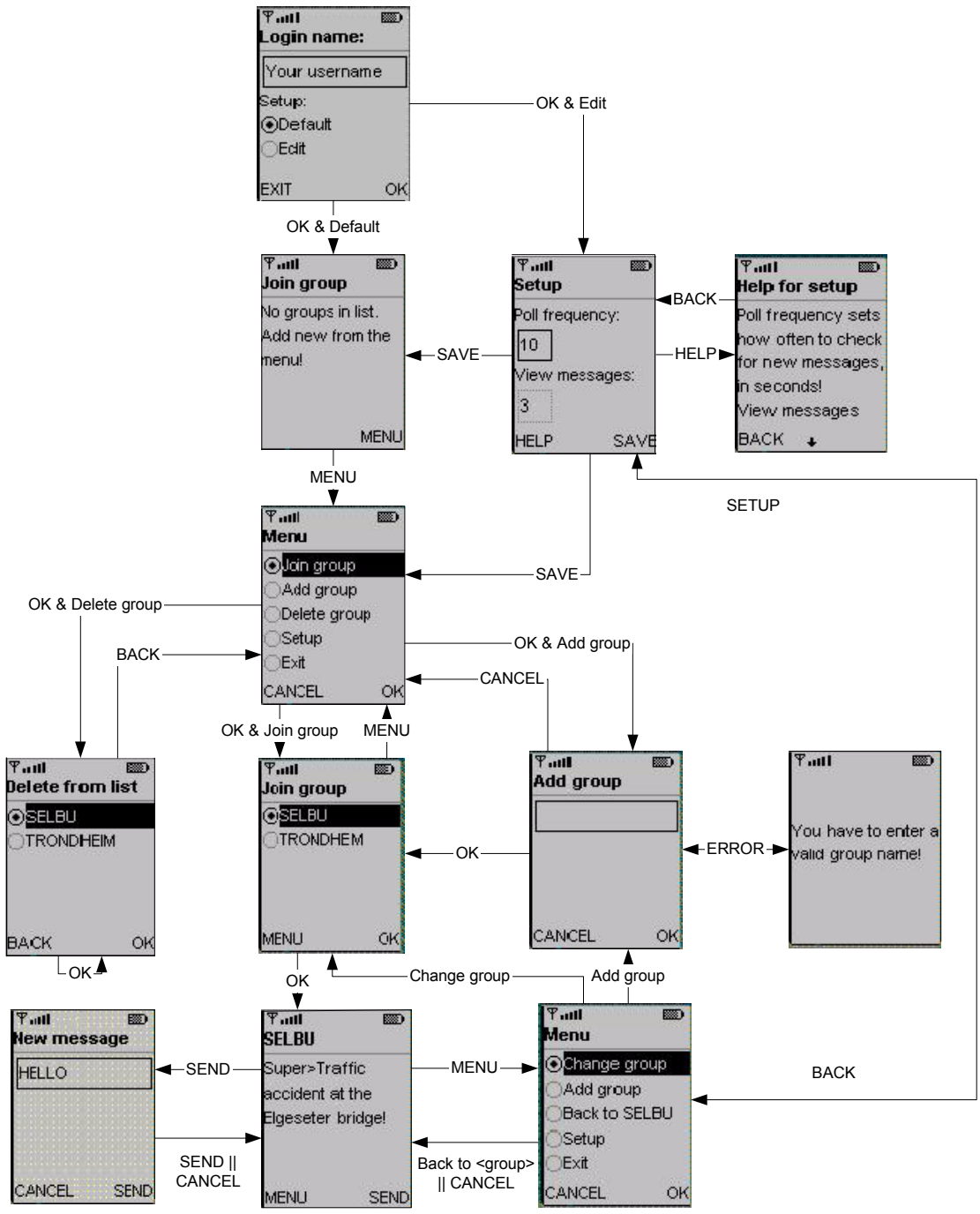


Figure 7-28 Screen flow for ITSSystem v1.0

In later versions, when private room is supported and the search functionality is made available in the JXTA proxy, additional fields and screens will be part of the system. The setup screen will have additional fields for choosing between private and public mode. Joining and editing of groups will be similar for private and public communication, but when the possibility for searching for groups and contacts is handled in the JXTA proxy, the screen for adding new groups will look somewhat different. In the current version, adding a group means adding it to the local group list, and when joining the group, it is checked whether the group already exist of not. Future version should change this functionality to a search pro-

cedure, where the user can choose to search for group or contact with the exact same name as the entered name, or to get a search result of groups or contacts with similar names.

In private mode the user will get a list over online contacts after joining a group, and the menu will include options for editing the contact list in addition to most of the existing options. There will also be a screen where received messages are printed, and the user gets the possibility of answering the message. Figure 7-29 illustrates the flow for a future version of ITSSystem. Some of the screens from the previous screen flow are left out because there are no changes.

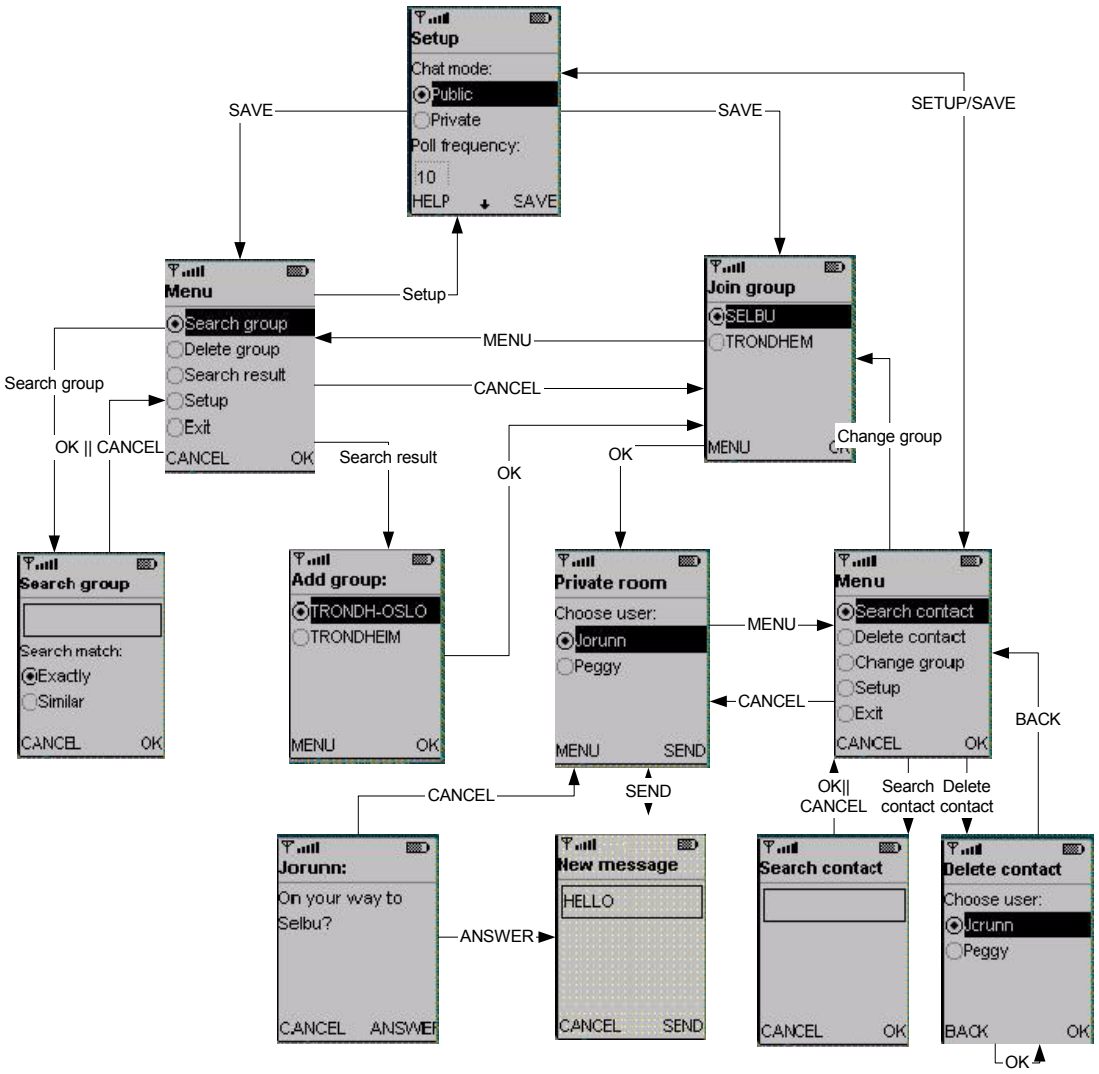


Figure 7-29 Screen flow for future version

The next subsection will discuss some implementation issues, and the screens illustrated in this section will help getting a better understanding of our solutions.

7.4.2 Error and exception handling

The prototype has different levels of error handling. This includes handling of external errors outside the control of the prototype, and internal errors like wrong parameter values and null pointer errors.

External errors is handled by usage of specific exceptions introduces in the prototype; DatabaseException and NetworkException. These are thrown if any critical error has occurred in communication with

the database or network. The ITSClient is left to handle the exception, but if the error is thrown upon initialization of the DatabaseClient or NetworkClient, the prototype is unable to continue and is therefore ended.

DatabaseException is thrown if the DatabaseClient is unable to initialize contact with the data repository, which in our design is the database included with MIDP (the Record Management System). If a user fails to contact the database, the application will not be able to run properly since it depends on the database to retrieve and store groups. If the ITSClient receives a DatabaseException at initialization, the user will get information about the error and the network connection will be closed before the application exits. This means that all ongoing activity will be cleared up before ending. DatabaseException will also be thrown when storing of data to the database fails. As with initialization, an error message will be displayed to the user, but the application will not exit this time.

NetworkException is thrown only if the NetworkClient cannot connect to the network, which is of critical importance to the application. An error message will be given to the user before the application clear up ongoing activity, and exits. This includes storing the group list to the database.

Internal errors include malformed parameters and occurrence of null pointer-objects. The ITSClient handles situations where the user leaves mandatory text fields empty, by alerting the user with an error message. The methods in NetworkClient checks all critical parameters whether they are set to null or not, and takes different actions according to this. In addition the NetworkClient always ensure that the client is never participating in more than one group, and that it is part of a group before sending a message. DatabaseClient will return null if no groups exists, and this is handled by the ITSClient by not displaying any groups and not give the user the possibility to click ok in the group list view. If no groups exist when closing the application, the DatabaseClient will receive null and then delete all records in the database concerning groups.

7.4.3 Parallelism

Several operations will occur at the same time in the application. The ITSClient must handle user input and possibly receive messages from the NetworkClient at the same time. The NetworkClient will have to respond to calls from the ITSClient, and at the same time poll the proxy at set intervals.

This parallelism is handled by use of threads. ITSClient handles the user interface, which is a thread on its own, awaiting input from the user. The input is caught and handled by a listener. This makes the ITSClient capable to receive messages from NetworkClient with ordinary call to the `receiveMessage`-method.

NetworkClient will at initialization initialize and a thread that is responsible for polling the network. This thread is unaffected from later method calls from ITSClient on NetworkClient.

Another solution to the NetworkClient thread is usage of a Timer that starts at certain time intervals. It might have been more suitable since a timer can be set to execute at the supplied time interval, regardless of the time it takes to handle the incoming message. On the other hand, timers is not always supported in the hardware, and therefore, and will therefore not be the best solution.

7.4.4 Minimal number of instances

Figure 7-13 illustrates the overview of the classes in the ITSSystem. The diagram states that only one ITSClient, one NetworkClient, and one DatabaseClient can exist in an application. To ensure this, both NetworkClient and DatabaseClient use factory methods to handle initialization of these objects. The factory method checks whether an instance already exists, and if so, it returns the existing instance of the object. If no instance exists, the object will be instantiated, and a reference will be kept for later retrieval.

7.4.5 Authentication

Mobile terminals are personal and are mostly used by only one person, the owner. As a result, username and password is not that important as it would be if the device where to be used by many persons.

A username is required in the prototype since this is used to identify the user in the ITSSystem. By now, the user types the username at start-up. Another solution could be to have the username, and alternatively a password, as an application property stored in the JAD-file accompanying the application jar-file. Using an application property allows you to customize a MIDlet suite for a particular customer (user). For example, as part of a servlet-controlled download process, the customer could be asked to enter an e-mail address and a preferred username. The process could then generate a random password and a custom version of the MIDlet suite, and send the username and password to the user by e-mail. At application start-up, the username and password has to be supplied, but afterwards it is up to the user to set if he wants to supply username and password at all login.

7.4.6 The search functionality

The search functionality of the prototype is implemented but not supported in the proxy server, and as a result nothing will happen if a user tries to search for a group. The thought is that the user, when adding a new group to his list of groups, should search for the group to obtain the id, if existing. The search can be for exact matches to the specified group name, or for similar group names. At exact search the user will get the group id in return. If the group does not exist, the group will be created when the user chooses to join the group.

Search for similar groups will result in a list of groups that the user can access from the group menu. The reason for not viewing the search result immediately after the search is because of the asynchronous nature of JXTA. When the list is viewed, the user can choose the most appropriate group to be added to the group list.

7.4.7 Network independence

The non-functional requirement NF-1, which demands network technology independence, is fulfilled by putting all the network functionality in one class; NetworkClient. None of the network specific classes are visible for the ITSClient. The data type ITSMessage is introduced to exchange information between the ITSClient and the NetworkClient. The application is also independent of how local data is retrieved and stored since all functionality for fetching stored information and storing information is hidden in the DatabaseClient.

To further ensure the independence, an exception hierarchy for the ITSSystem is introduced. The superclass, ITSException, is abstract meaning that it can only be instantiated by instantiating one of its subclasses. ITSException has two subclasses NetworkException and DatabaseException. These exceptions makes the ITSClient able to make decision on background of the information, notify the user, and end the application if a critical exception occurs. As a result, the ITSClient is not concerned with network-dependent or database-dependent exceptions.

The only drawback of this version of the ITSSystem when it comes to network dependence, is that a user must be able to supply an IP address and port to a proxy he wants the application to connect to. This dependency will hopefully be removed in the future, when the JXTA for J2ME packages itself takes responsibility to find the nearest proxy and connect to it.

7.4.8 Group membership

A user of ITSSystem can only be member of one group at a time. There is no limit in JXTA for J2ME on how many groups a user can be member of. The reason for this limitation in ITSSystem is based on

usability issues. If a user had the possibility to be member of more than one group, the program had to be much more complex, leading to lower usability.

A user could also, in principle, be able too be in both private and public communication mode at the same time. Again, this would require to much from the user, as the program would have become even more complex and hard to follow.

7.4.9 User interface

One of the challenges with making programs for mobile terminals is the limited input possibility. The developers of such programs must choose between a solution which is easy to use, but requires several screen and clicking, or a solution which is not obvious to understand at first, but does not require so much effort when first understood.

The ITSClient should first of all illustrate the concepts of J2ME CLDC/MIDP and the possibilities with JXTA for J2ME. When the requirements for the main functionality were fulfilled, the usability was considered. After carrying out a couple of user test, some changes needed to be done to the system. If this was a real system, and not a prototype meant to demonstrate and understand the technology of J2ME, the user tests would have been taken more under consideration. Section 7.6.5 will summarize the result from the user tests, and alternative solutions will be presented.

7.5 ITSSystem in the future

In this version of the ITSSystem, the service is actually a chatting service that could be used in other occasions as well. By using location based technology the service could be more specific for the domain it was actually intended for. Many improvements must have been made to the system to make it really useful, and this chapter will make some suggestions for future versions of the system. Some of the solutions are not yet possible because of missing or incomplete technology.

7.5.1 Location based technology

Ideally, the application should be able to figure out where the users are situated and join the appropriate group based on this information. If more than one group is available, the user should be asked to choose between the given groups for the area. In this way the user should be able to use the service without knowing the location.

Such a solution could be achieved by use of Global Positioning System (GPS), or other positioning systems. Next generation of mobile communication, like UMTS, will include positioning.

7.5.2 Speech

Another improvement of the system would have been if the user could control the application by use of speech. In the current version, the driver is dependent on passengers to send and retrieve information. With introduction of speech recognition and speech control, a driver on his own would be able to use the prototype. This could be very useful for truck drivers, for instance, who drives mostly on their own.

Control by speech should also include that the receiving message could be presented from the terminal by speech and not only by text. The user should be able to choose type of control and communication, either speech or text.

7.5.3 Maps

To further improve the service, the user should have the possibility to send and receive maps over a specific location. Maps should also be available on different servers on the network. J2ME already supports

downloading of png-images using HTTP, so this should be a minor addition that adds great value to the service.

7.6 Testing

Different standards exist for testing of application. These states the procedure for testing, i.e. what to go through when testing. Other focuses on the security requirement for the application, and states demands for the development, testing, acceptance, management and maintenance according to the level of security.

IEEE [IEEE] has several standards, including standards for testing and test documentation. The test documentation standard, IEEE Standard for Software Test Documentation from 1998 (829-1998), is available online for a fee. It describes a set of software test documents that will provide a common form of reference and increase the manageability of testing. The 1991 version is available at [829-1991].

This test document will not follow any standard, as it would be too extensive to such a small application. First it will provide a description of the testbed used in the testing, and then presents the test specification test results and comments. The test specification covers all the functional requirements of the prototype, even though only part of the requirements is implemented.

7.6.1 Functionality and implementation testing

The functional testing should verify the functional design against the functional requirements. The test should also detect possible internal inconsistencies and errors in the functional design and validate the functional design against the users needs. In practise the functionality testing is executed for the first time in the implementation, and this was the case for this system development. As a result, the design diagrams were somewhat changed during the implementation.

Implementation testing can be performed by comparing the implementation with the functional design, and by testing the non-functional requirements. The implementation testing was executed during the implementation of the different versions, and after each version was completed. In this way, we did not continue with the implementation of a new version until the previous were functioning. Since version 1.0 was the last version that was implemented, a complete system testing was performed when finished.

7.6.2 Testbed

The prototype was implemented using J2ME Wireless Toolkit [J2MEwtk] and the supplied emulator in this toolkit. It was also tried tested on an iPaq using a WLAN card, both with the Jeode virtual machine from Insignia with the me4se-emulator, and by usage of the j9 virtual machine from IBM. Unfortunately there where some problems with both platforms. Using the Jeode with me4se the application went down since it could not establish network connection with the proxy. The j9 virtual machine had no desire of running at all. An introduction to the tools used in the implementation and testing of the prototype is given in Appendix E.

When it comes to phones, the prototype was tested on a Nokia Communicator 9210 with a beta implementation of J2ME, and as with the Jeode solution, it failed trying to connect to the network.

The testbed presented in this document is the wireless toolkit with supplied emulators. If some of the solutions for the iPaq had been more successful, these as well would be a testbed.

7.6.3 Test specification and results

A detailed test plan for version 1.0 was made with test cases and the desired results. The testing were separated in five parts according to the main functionality of the ITSSystem:

1. Login
2. Setup
3. Group overview
4. Group list editing
5. Group chat (public mode)
6. Contact overview (private mode)
7. Contact list editing (private mode)

The test specification includes a number for each test with by it can be referenced later, a test case description or scenario, the desired result, and the result of the test using the testbed described in chapter 7.6.2. Tests can be passed (OK), not passed (NOK) or not tested (NT).

Login

The starting point for the test cases assosiated with login is at application launch. Table 7-2 shows the test plan for login and authentication.

No.	Test case	Desired result	Result
T-1.1	Log in with correct username and choose default setup	The "Choose group" screen appears with a list over the groups, if any.	OK
T-1.2	Log in with correct username and choose to edit setup	The setup screen appears	OK
T-1.3	Log in with wrong username	An error message stating "Incorrect username!" appears.	OK
T-1.4	Click the EXIT button	The application closes	OK

Table 7-2 Test specification and result for login

Test T-1.4 has as result that the application will close upon hitting the EXIT button. Closing implies that the connection to the database and the network is closes for all open connections, and that all groups and contacts are saved to the database.

Setup

Testing the setup requires the user to start the application and choose "edit setup" at start-up. This will bring the user to the setup-screen and he will be presented to several possibilities to regulate the running of the application. The test cases presented here have starting point from this screen, and is presented in Table 7-3.

No,	Test case	Desired Result	Result
T-2.1	Use the default values in the setup screen.	The ITSClient connects to the relay and the user is presented for the group overview, a list of groups, if any exists.	OK
T-2.2	Change the value of the relay host and/or relay port.	The application check that it is a valid IP-address and that the port number is above 1024, and then connects to the specified relay. The group overview is displayed for the user, a list of groups, if any exists.	NOK

Table 7-3 Test specification and result for setup

No,	Test case	Desired Result	Result
T-2.3	Change the polling frequency.	Checks whether the set polling frequency is bigger than 0, then set the new pollfrequency before following the same result as for T-2.1..	OK
T-2.4	Change the value of "number of messages".	Check whether the value is bigger than 0. Only the selected number of messages are viewed at once when the user is chatting in public mode. Follows the result as for T-2.1.	OK
T-2.5	Click the EXIT button	The application closes.	OK
T-2.6	Choose public mode as the communication mode.	The application will be started for public mode communication. Follows the same result as T-2.1.	OK
T-2.7	Choose private mode as the communication mode.	The application will be started for private mode communication. Follows the same result as for T-2.1	NOK
T-2.8	Change the polling frequency to 0 or an empty string.	An error message is displayed, and the user will return to the setup screen.	OK
T-2.9	Change the value of "number of messages" to 0 or an empty string	An error message is displayed, and the user will return to the setup screen.	OK
T-2.10	Supply a non-valid IP-address in the relay-field, or a port number that is smaller or equal to 1024.	An error message is displayed, and the user will return to the setup screen.	NT

Table 7-3 Test specification and result for setup

In the setup, the user gets the possibility to edit the communication mode, relay host and port, number of messages to view in public mode, and the poll frequency. Version 1.0 of the system does not support the possibility to private communication, but "Private" is anyway shown as a choice. If the user selects "Private", an error message will appear. Setting the relay host and port is no supported in this version, so changes to these fields will not be handled. In other words, the default values for relay and port is used upon connecting to the network.

If a user enters the setup while still member of a group, he will not leave the group as long as relay-address and port is unchanged. In the current version, this is not implemented, so the user never leaves the group he is member of.

Group overview

Testing the group overview requires the user to start the application and log in, with or without changes to the setup. This will bring the user to a screen showing the different groups in the database, if any exists. The test cases presented here all have starting point from this screen, and all tests must be completed independent of the others. The test cases and results are presented in Table 7-4.

No.	Test case	Desired result	Result
T-3.1	Select a group in the list when in public mode.	If no groups are available, this choice will not be possible. Otherwise, the user joins the selected group, and the screen for public communication appears.	OK
T-3.2	Select a group in the list when in private mode.	If no groups are visible, this choice will not be possible. Otherwise, the user joins the selected group, and the user will be presented with a list of his contacts, if any.	NT
T-3.3	View the group overview screen with no groups present.	No groups are listed and the OK button is not visible.	OK
T-3.4	Click the MENU button	The group menu appears.	OK

Table 7-4 Test specification and result for group overview

Test number T-3.2 is not tested since private communication is not supported in this version of the prototype. The group menu reference in T-3.4 is further investigated in the next subsection.

Group list editing

Testing the group list editing requires the user to start the application and log in, with or without changes to the setup. This will bring the user to a screen showing the different group sin the database, if any exist. In this screen, tap the MENU button. The test cases presented here have this screen as starting point.

No.	Test case	Desired result	Result
T-4.1	Select “Setup” from the menu and click OK.	Enter the “Setup” screen.	OK
T-4.2	Select “Add group” from the menu and click OK.	Enter the screen for adding a new group.	OK
T-4.3	Select “Delete group” from the menu and click OK.	Get a list over groups to delete, if any.	OK
T-4.4	Select “Exit” from the menu and click OK.	The application closes.	OK
T-4.5	In the add group view, write a new group name and select “Exactly” search and then click OK.	If the group does not exist, it will be added to the group list, and a search for the a group that exactly matches the groupname supplied is started. If the group exist, nothing will happen. The application returns to the group overview screen.	NOK
T-4.6	In the add group view, write a new group name and select “Similar” and then click OK.	If the group does not exist, and a search for all groups that contain the given name as a substring is started. If the group exist, nothing will happen. The application returns to the group overview screen.	NOK

Table 7-5 Test specification and result for group editing

No.	Test case	Desired result	Result
T-4.7	Write an existing group name in the text box, select "Exactly" or "Similar" and click OK.	No new group is added, and no search is started since the group already exists. The user returns to the group menu.	OK
T-4.8	Do not write a group name in the text box and click OK.	No group is added to the list. An error message is displayed to the user, and the user returns afterwards to the add group screen.	OK
T-4.9	Click CANCEL in the add new group screen.	No group is added to the list. The user is redirected to the menu screen.	OK
T-4.10	Select "View search result" from the menu and click OK.	The user is presented with a list of result from the most current search with the "Similar" option.	NT
T-4.11	Select a group in the "View search result" screen and clik OK.	The group is added to the group list and the user is redirected to the group editing menu.	NT
T-4.12	Choose a group in the list of groups to delete and click OK.	The group will be deleted from the database, and the list of groups that can be deleted is shown to make the user able to delete more groups.	OK
T-4.13	Click the CANCEL button in the list of groups to delete screen.	The selected group are not deleted and the user is redirected to the group menu screen.	OK
T-4.14	Delete all groups and view the delete group screen.	No groups are listed and the OK button is not shown.	OK

Table 7-5 Test specification and result for group editing

Test T-4.5 and T-4.6 did not pass. The functionality is implemented in the prototype, but the JXTA for J2ME proxy does not support search yet and so it was impossible to test it sufficiently. As a result, the prototype is not tested for T-4.10 and T-4.11 since this functionality is not implemented as a result of the above observation.

The alternative implementation of T-4.5 and T-4.6 is documented and tested in Table 7-6.

No.	Test case	Desired result	Result
T-4.15	Write the name of the group in the text box in the "Add new" screen, and select "Exactly".	If the group does not exist, it is added to the local hashtable and the group editing menu reappears.	OK
T-4.16	Write the name of the group in the text box in the "Add new" screen, and select "Similar".	Same result as for T-4.15.	OK

Table 7-6 Alternativ test specification for T-4.5 and T-4.6

Group chat (public mode)

Testing the group chat requires the user to start the application and log in public mode, with or without further changes to the setup. In the group view screen, the user will have to select one group to join,

which will bring him to the group chat room. The test cases presented here have this screen as starting point, and is summarized in Table 7-7.

No.	Test case	Desired result	Result
T-5.1	Click NEW in the group chat screen.	The "New Message" screen with a text box and the possibility to click SEND and CANCEL appears.	OK
T-5.2	Click MENU in the group chat screen.	A menu with the options "Change group", "Setup" and "Exit" appears.	OK
T-5.3	Write a message and click SEND	The name of the user and the message will appear on the screen. The maximum number of messages printed to the screen depends on the value set in setup or the default value (3). The message is sent to the group, and the application returns to the contact list view.	OK
T-5.4	Receive a message from another user in the same group (for example send from another emulator)	The name of the other user (the sender) and the message will appear on the screen.	OK
T-5.5	Click OK in the group chat with a empty message-string.	An error message is displayed, and the user returns to the message send screen.	OK
T-5.6	Write a message that contains more than 256 characters.	Not possible. The user will not be allowed to type more characters than 256.	OK
T-5.7	Write a message and then click CANCEL.	The public communication screen appears without sending the message just written.	OK
T-5.8	Select "Setup" in the menu and click OK.	The user enters the "Setup" screen, but does not leave the group. The group is only left if the user alters variables in relation to the network, like relay-address and port.	NOK
T-5.9	Select "Change group" in the menu and click OK.	The user leaves the group and enters the group list screen that lists the available groups, if any.	OK
T-5.10	Select "Exit" in the menu and click OK.	The application closes.	OK
T-5.11	Click BACK when in the menu.	The user is still participating in the same group, and the screen for public communication appears.	OK

Table 7-7 Test Specification and results for group chat (public mode)

T-5.8 was not passed since the prototype never leaves a group upon exiting the "Setup" screen since the variables for relay address and port is never checked for changes. As of today, the user will always be member of the group until he actively leaves the group.

Contact overview (private mode)

Testing the contact overview requires the user to start the application and log in private mode, with or without other changes to the setup. From the group overview, the user must select a group to join, and

will upon joining a group be presented for the contact list. The test cases presented in Table 7-8 have this as their starting point.

No.	Test case	Desired result	Result
T-6.1	No contacts in the list	The user is only capable of entering the menu.	NT
T-6.2	Select a contact	A screen for writing a message to the chosen contact appears.	NT
T-6.3	Write a non-empty message in the box that appears upon choosing a contact, and click OK.	The message is sent to the contact, and the user returns to the contact list view.	NT
T-6.4	Write an empty message in the box that appears upon choosing a contact, and click OK.	An error message is displayed to the user, and the user returns to the message screen.	NT
T-6.5	Write a message that contains more than 256 characters.	Not possible. The user will not be allowed to tpye more characters than 256.	NT
T-6.6	Write an message, non-empty or empty, and click CANCEL in the message-box.	No message is sent, and the user returns to the contact list screen.	NT
T-6.7	Click the MENU button	The user is confronted with the contact list menu.	NT
T-6.8	Recieve a message from a contact when viewing the contact list.	The message will be displayed in the recieve message view.	NT
T-6.9	In the recieve message view, choose cancel.	The contact list will pop up.	NT
T-6.10	In the recieve message view, choose reply.	The user will be presented for a screen that gives him the possibility to write a message, as for T-6.3 to T-6.6.	NT
T-6.11	Receive a message from a contact when writing or recieve a message to/form another user.	The message is stored for later retrieval, that is to say, when the user leaves the current screen and return to the contact list view.	NT

Table 7-8 Test specification and result for contact overview (public mode)

Private mode is not implemented in the current version of the prototype, and so the above tests have not been performed.

Contact list menu (private mode)

Testing the contact list menu requires the user to start the application and log in private mode, with or without other changes to the setup. From the group overview, the user must select a group to join, and will upon joining a group be presented for the contact list. In the contact list view, the user must click the MENU button. The test cases is presented in Table 7-9 have this as their starting point.

No.	Test case	Desired result	Result
T-7.1	Select “Setup” from the menu and click OK	Same result as T-5.8.	NT
T-7.2	Select “Add contact” from the menu and click OK.	Enters the screen for adding a new contact.	NT

Table 7-9 Test specification and result for contact editing (private mode)

No.	Test case	Desired result	Result
T-7.3	Select “Delete contact” from the menu and click OK.	Enters the screen for deleting a contact.	NT
T-7.4	Select “Exit” from the menu and click OK.	The application exits.	NT
T-7.5	In the add grop view, write the name of the new contact.	If the contact not already exist in the contact list, it will be added to the contact list and a search for the contact is started. If the contact exist, show an error message.	NT
T-7.6	Do not write a name for the contact (empty string) and click OK.	No group is added. An error message is displayed to the user, and the user returns to the add contact screen.	NT
T-7.7	Click CANCEL in the add new contact screen.	No group is added. The user is redirected to the menu screen.	NT
T-7.8	Select DELETE in the menu, and then choose one of the contact in the list that is displayed.	The contact will be deleted from the list, and the user will be redirected to the contact list editing screen.	NT
T-7.9	Click the CANCEL button in the delete contact screen and click OK.	The selected group is not deleted, and the user is redirected to the contact list menu screen.	NT
T-7.10	Delete all contacts and view the delete contacts screen	No groups are listed, and the OK button is not shown.	NT

Table 7-9 Test specification and result for contact editing (private mode)

7.6.4 User test tasks

Another important part of the testing is the user test. This involves inviting a number of potential users to try out the program and valuate the user interface and the user-friendliness. Nine persons were invited to be testers for the program, four male and five female. Three of the persons had no experience with similar programs and little technical knowledge. The rest were telematics- and computer science students from NTNU.

The test persons were first shortly introduced to the service and then they got a list of tasks to fulfil. During the user test, the users comment on difficulties and came with suggestions to improvement. Some of the problems they had were related to the user interface on the specific emulator. Other problems came because an emulator combines the input possibilities of a mobile device with the input devices (like mouse and keyboard) of a personal computer. These types of problems are not taken into consideration in this section.

None of the test persons were given a user manual before the testing started. The reason for this is that in real life users usually want to try out a new program by testing and failing, and the user manual is mostly used as a complement. Independent of this a user manual is written and can be found in appendix D.

The tasks that the user were confronted with is listed below:

1. Start ITSSystem and log in using "SUPER" as username.
2. Add the group "Selbu" to the group list and join this group.
3. Send some messages. (Communicate with other users.)
4. Add the group "Trondheim" to the group list and join this group.
5. Send some message. (Communicate with other users.)
6. Set the number of messages to be viewed to 2 and the polling frequency to 3 in setup

7. Continue to communicate in group "Trondheim"
8. Change group to Selbu
9. Delete the groups "Trondheim" and "Selbu"
10. Exit the program.

The problems that the users had depended partly on the experience they had with similar programs (like chat programs or instant messaging programs) and with mobile terminals. The result of the test is presented in the next sub chapter.

7.6.5 User test result

For the test persons with little technical background and no experience with similar programs the main challenge was to understand the screen flow in the program and how to return to a specific screen. The menus specially confused them. They were specially confused by the menus. Since both were called "menu", they expected that the menus were the same, and had some problems with understanding why some options in the menu had changed. Other persons with more technical background suggested also that the menus should be similar or almost similar. For instance they meant that "Delete group" should be an option for both of the menus.

Many of the test persons stated that they wanted confirmations when they edited something, like the setup or group list. Specially after saving the setup they would like that a confirmation screen appeared, assuring them that the setup was changed. Some of the users wanted the same thing to happen when editing the group list, while others were satisfied with the current solution where the group list were updated after adding or deleting a group. Since the ITSClient already have implemented a method for viewing error-messages and confirmation, this would not have be a time consuming improvement of the system.

Another suggestion from some of the test persons was to arrange the editing and joining of groups in a different way. When the group list appears the user should select one of the groups and the next screen will give the user the possibility of either join or delete that specific group. The reason for not choosing this solution is that the primary goal of this service is to join a group and communicate in that specific group. If the user must first select a group and then a new screen with the possibility to join that group appears, the user must go through an extra screen before he can start to communicate.

Other comments were concerning the names of the different buttons. For instant if a user wants to write a new message, the "SEND"-button must be pressed. Some of the users did not think this was obvious, and wanted a more informative name for the button, like "NEW MESSAGE" and "SEND MESSAGE". The problems with this is that the text is to long.

The last suggestion that were made from some of the users was to let the user decide whether to leave the group or not when editing the setup or adding a new group for instance.

The user test was a very effective way of testing the usability of the program. Nothing was changed after these final user tests, but in a real situation this would have been taken into consideration before the final result of a program was released.

7.7 Summary

This chapter has presented the design and implementation of the ITSSystem, a prototype developed to show some of the features of J2ME MIDP and JXTA for J2ME.

The ITSSystem is a first generation location based system, where users being in a specific area are able to communicate and exchange information related to the users location. Typical areas are big cities or distances where users are out travelling, for example driving from Oslo to Trondheim.

Both functional and non-functional requirements were stated and a design where made. All the necessary components for a working system was included, but only part of the system were implemented; the public room. In public room users can participate in a group according to location, and messages from one group member is sent to all members of the group. The user is able to join groups, add or delete groups in his private list and edit setup to control how many messages that are visible at a time in a conversation and how often to poll the proxy server.

Some implementation issues were discussed, like how the implemented prototype meets some of the non-functional requirements, how errors and exception are handled, and some limitations of the prototype. Also some thoughts for the future for further improve the value of the system where mentioned. Like controlling the application by speech, deciding the location based on location based technology and alternative network solutions.

The last part of the chapter presents the test specification and the test result for the implemented prototype using the wireless toolkit from Sun, and the results of the user test that where carried out.

Chapter 8

Discussion

"The important thing in science is not so much to obtain new facts as to discover new ways of thinking about them."
Sir William Bragg

A number of subjects have been covered in this master thesis, and what would be interesting is to see how they all relate to each other. J2ME MIDP has shown that it has several capabilities to support network connection, both client/server and peer-to-peer technologies like Jini and JXTA. Even though Jini and JXTA share many features, they have several differences both among themselves and in relation to other technologies.

This chapter will provide a comparison of Jini and JXTA, both the main architecture and the solution for embedded devices. Further it will look at different scenarios on how to make the two technologies interoperate, before moving to see how they part from Web Services and Microsoft .NET. As part of the assigment a prototype was developed to gain experience with JXTA, and a discussion of some areas of the protype is provided.

Jini and JXTA might be used in many project, and the second last section propose several scenarios on how they can be used in the AMIGOS project, before moving on to some future scenarios.

8.1 Comparing Jini and JXTA

This section will first provide an overview over the main similarities and differences of Jini and JXTA, and then point out the advantages of each technology, taken that Java is already chosen as programming language.

8.1.1 Differences

Jini Network Technology and JXTA have a lot in common, but they were designed with different assumptions. The difference between Jini and JXTA can be summarized in respect to three areas:

- **The platform required**
- **Communication**
- **Location of services**

The main different between Jini and JXTA is in the *required platform*; Jini assumes the present of Java in the network, while JXTA have no assumption when it comes to programming language. Jini requires Java to publish services. If a device does not support Java, a special proxy service "in Java" is necessary, and the developer will have to implement the bridge. The reference implementation of Jini is also dependent upon TCP/IP as transport protocol. JXTA is language independent and platform independent, since it is entirely protocol-based, thereby allowing endpoints to be implemented in a variety of ways. The JXTA

reference implementation is in Java, but JXTA is currently being developed in C, Perl, Smalltalk and Python.

Communication is achieved differently for Jini and JXTA. In Jini services communicate with the lookup service using RMI, which transfer serialized objects, proxies, to clients that ask for the service. The proxies can communicate with the services using any protocol. JXTA communicates using XML or binary messages (advertisements), that are transfered using the JXTA protocols. The protocols are independent on transport protocol. Essentially, the difference lies in that Jini networks consists of mobile objects that are transfered, while JXTA networks consists of XML or binary advertisements.

Location of services is the last area of differences covered in this section. Jini uses a central broker, while JXTA does not. Jini Lookup Service is not really "central" in any traditional sense since many may be available. Contents on the Lookup Services may not overlap, and redundant Lookup Services can be equally reachable, and so making them naturally fault tolerant and easily load balanced when using automatic discovery. In the Jini vs. JXTA sense, the Lookup Service is more "centralized" because the lookup info is often located on a limited number of Lookup Services. The Lookup Service is itself discovered, not pre known. In JXTA, services are made public through advertisements, and devices have to search for the services among all the peers in the network, i.e. there is no central broker to contact. In connection to this, it can be seen that while JXTA seek to implement a pure peer-to-peer solution, Jini is with its lookup service an hybrid peer-to-peer solution.

8.1.2 Similarities

Jini and JXTA share a number of similarities, despite the differences covered above. These similarities are:

- **Protocol Frameworks**
- **Services**
- **Discovery**
- **Spontaneous networking and recovery from failure**

Jini and JXTA are both *Protocol Framworks* for creating peer-to-peer structures. A framework involve the structuring of processes, while protocols involve the transfer of information. JXTA consist of six protocols that handles all communication, while Jini has three protocols that handle discovery and join of services.

Services are an essential idea in both specifications. Both address the problem of presenting "services" as available on a network or across networks. A user should be able to access services at any time, from any device, and independent of location.

Jini and JXTA both use *discovery* protocols to find services. Discovery in Jini consists of discovering the Lookup Service, while in JXTA discovery is general, i.e. used to find services and other entities in the network directly. But there is no limitation in the Jini architecture that keeps it from running lookup services anywhere, and so approximating JXTA and pure peer-to-peer architecture.

Spontaneous networking and recovery from failure is solved in both network specifications. One of the key ideas behind Jini is spontaneous networking, and another is recovery from failure. The goals are achieved through leasing and code mobility. The leasing mechanism in Jini is flexible and necessary in ad hoc networks and the mobile code can move securely from machine to machine on a network. In JXTA every advertisement is valid and available on the network in a specific, pre-defined amount of time specified in the advertisement. The advertisements floating around in the network provides the spontaneous networking since devices can discover services through the discovery protocol.

The primary criticism of JXTA has been that it essentially is a renaming and repacking of Jini. JXTA is promoted by Sun today in much the same way as Jini was. Jini was a first step at a distributed-computing standard, but never took off because it works only with devices that run Java. It has been claimed that the step towards JXTA is a move by Sun to try to gain acceptance of Jini that has not up to now materialized as they had hoped. On the other hand, Sun maybe realized that Java-dependency was not the right solution, and that a platform and language independent solution was more appropriate. JXTA may stand a better chance of success than Jini because it works with non-Java applications. Anyway, the two solutions can be used for different purposes, dependent upon what will suite an application best.

Many are wondering whether the two technologies will converge in the future because of the similarities. But the fundamental difference of universality via XML or Java mobile code may be enough to keep the two technologies in the market until one or the other technology becomes "obsolete".

8.1.3 Advantages and disadvantages

Despite the significant differences, the technologies seem very similar and one can envision them converging, or one taking precedence over the other. In connection to this, the different outcome dependens upon whether one is already committed to Java or not. If one is already committed to a Java platform, the differences decrease. The two first differences above will become insignificant, and the third is adjustable in Jini to essentially mirror a JXTA system. In other words, the major difference is usage of Java mobile code in the Jini protocols or the XML protocols of JXTA. Without adjustment to the location of services, the difference also includes whether one wants a centralized lookup of services or not. Both solutions are still discovery processes. Another thing that has to be mentioned is that the reference implementation of Jini depends upon RMI to transfer java data objects between Jini services. Taking this into account, it creates another dependency since all devices running Java must have RMI support, or they must use the surrogate architecture.

The rest of this section will assume that Java has been chosen as programming language, and that the reference implementation of the specifications is used. In relation to these choices one can regard the strengt of both Jini and JXTA, and the advantages and disadvantage are discussed below:

- **Efficiency** (both transfer and cpu): It is shown in several studies that any ASCII based protocol is inherently less efficient than a binary one. JXTA is a verbose ASCII protocol, and will so be even less efficient. In addition, JXTA depends upon generation and parsing of XML, which is very resource consuming. Compare this to the object serialization in Jini, which requires minor resources.
- **Operations on/by Type**: Jini uses Java objects, which enables asking for things by type, and to manipulate full Java objects. The benefits are the same as one expect from type checking, inheritance and polymorphism when working with objects rather than text. JXTA operates on XML or binary messages that cannot hold a state in the same way as an object, only textual information.
- **Interoperability**: JXTA makes is possible to communicate with non-Java participants, while the core Jini only can talk to entities in the network with Java-support. For Jini clients to talk to non-Java participants, the non-Java participants must use the surrogate architecture.
- **Discovery mechanisms**: The discovery mechanism of JXTA is more standardized than in the Jini and RMI environment. For example, services can be accessed despite of firewalls, although RMI has a proxy mechanism called RMIProxy that might help in these cases. JXTA rendezvous hosts can though also be created in Jini, but there is no standard for it.
- **Communication layers**: The communication layers in JXTA are abstracted and runtime bound, and so addresses some of the pitfalls you might encounter with Jini when crossing network, device, and platform boundaries.
- **Flexibility**: XML gives more flexibility than object serialization. Some disagree with this point, arguing that Java is a very dynamic and flexible language, and no example has been shown for something in XML that cannot be done in Java if one assumes that XML is used in communication between Java systems.

- **Complex**: JXTA might seem overwhelming with many concept and protocols to keep track of. Jini appears simple with its clients, services and lookup servers.
- **Debugging**: Debugging is easier in JXTA than in Jini since developers are able to look at a JXTA stream and understand what is going on. To debug Jini, or more precisely RMI, special debugging tools are needed to read the objects being passed back and forth, and peek into what RMI is doing under the covers. The debugging tools often use the Java reflection library, and a developer can use the library directly if he wishes. The Java reflection library API represents, or reflects, the classes, interfaces, and objects in the current Java Virtual Machine. It can be used to fetch information about a class' fields, methods, constructors, superclasses and all other information in realtion to the object.

8.2 Jini and JXTA architectures for embedded devices

Jini and JXTA have chosen different ways to solve how embedded devices can connect to the Jini and JXTA network respectively. Jini has developed the Surrogate Architecture while JXTA has several project targeting different devices, among other the JXTA for J2ME. The most interesting case in relation to this master thesis is to compare the surrogate architecture to the JXTA for J2ME project in the JXTA community.

Jini Surrogate Architecture and JXTA for J2ME cannot be compared directly since the surrogate architecture is a specification of a framework, while JXTA for J2ME is an implementation of JXTA aimed at embedded devices supporting J2ME CLDC and CDC. JXTA for J2ME provides an architecture that can be compared to the Jini surrogate architecture, but the architecture is based on the network capabilities of J2ME MIDP.

As a result of this fundamental difference, the discussion below has to take this into consideration. The section comparing the differences and similarities of Jini surrogate architecture and JXTA for J2ME takes the overall solution of both technologies into account, but assume that the devices using the architecture might support either J2ME CLDC or CDC. The section will first consider the differences and similarities, and then move on to the advantages and disadvantages of the two specifications.

8.2.1 Differences

The difference between Jini Surrogate Architecture and JXTA for J2ME is the same as discussed for Jini and JXTA above. In addition, several other differences are introduced in relation to the architectures for embedded devices.

- **Specification**
- **Target devices**
- **Availability**

The main difference between the Jini Surrogate Architecture and JXTA for J2ME comes in relation to whether they are *specifications* or not. While the surrogate architecture is a specification, JXTA for J2ME is an implementation of JXTA aimed at embedded devices supporting J2ME CLDC or CDC. But both define an architecture that are based on the capabilities of the involved devices.

The *target devices* of Jini Surrogate Architecture and JXTA for J2ME differs in more than one way. The Jini Surrogate Architecture is a common platform used by a range of devices, from sensors to applications written in Java or another language. JXTA does not have any common platform to support embedded devices, and provides a variety of different packages and solutions dependent on the device and its capabilities. If a device supports XML and is able to listen for incoming requests it uses the JXTA protocols, otherwise specialized solutions are needed. One of these packages is JXTA for J2ME that require support for J2ME CLDC or CDC, and uses HTTP to contact a relay that acts on its behalf on the network.

Availability can be defined to what degree an embedded device will be able to participate in the network. In Jini a host capable machine must be set up specific as a surrogate host for a device, or a number of devices. JXTA solves the availability issue by making each rendezvous peer able to be a relay for embedded devices.

8.2.2 Similarities

Despite the differences, the two architectures for embedded devices share some common properties.

- **Objective**
- **Network independence**
- **Dependence upon a relay**

The *objective* of Jini and JXTA is to support access to services from any device, and the Jini surrogate architecture and JXTA for J2ME extends the area of availability of the technology. As a result, they fulfil the goals of both Jini and JXTA.

Network independence is a central goal both in Jini and JXTA, even if dependencies is introduced in the implementations of the specification. Jini Surrogate Architecture does not specify how the devices interconnect with the surrogate host. This is left to different interconnect specifications as for example the IPInterconnect and WirlessDevice Project (Bluetooth interconnect). JXTA for J2ME aims at supporting different interconnect protocols when they are available as the lowest common denominator. As of today this is HTTP, but the packages are built to make a change to other protocols easy as long as the new protocols are supported in the Generic Connection Framework.

The architectures are both *dependent upon a relay* to act on the devices itself. The Jini surrogate host and the JXTA relay must both be pre known by the device as of today since there are no mechanisms that supports discovery of the relays. In addition, both Jini and JXTA aims for independence of the relay. PsiNaptic, a communication technology company, has removed the dependency on RMI and made a lite version of Jini available for embedded devices, and as a result there is no need for a relay. JXTA also has a goal to limit the dependency upon a relay server, first by having a discovery mechanism to discover relays dynamically and when the abilities in J2ME CLDC/MIDP allow it; implement a version of JXTA for J2ME that communicate directly with the JXTA community.

8.2.3 Advantages and disadvantages

The assumption for discussing the advantages and disadvantages of the Jini Surrogate Architecture and JXTA for J2ME is that the device must be java-enabled with the J2ME MIDP library. It also assume use of the reference implementation of the Jini Surrogate Architecture in addition to the IPInterconnect reference implementation when it comes to Jini. In the JXTA case is uses the current implementation of JXTA for J2ME. All these reference implementations uses the Java programming language. According to this, the advantages and disadvantages are:

- **Complexity**: Jini surrogate architecture is more extensive than JXTA since it is a specification that is dependent upon interconnect specifications to be able to connect the devices to the Jini network. The developer must program both the surrogate and the device to communicate with the surrogate, which are two different packages. The relay in JXTA is part of the JXTA network, and operate using the binary version of the JXTA protocols, making the developer able to concentrate on a small and simple API at the device.
- **Availability of implementation**: There is available a reference implementation for Jini Surrogate Architecture, and an IPInterconnect reference implementation. JXTA for J2ME is an implementation, and is available with minor shortcomings in the relay like lack of search-capability.
- **Transport protocol dependence**: The available implementations in relation to Jini Surrogate Architecture makes the devices dependent upon IP, which might not be implemented by all J2ME

enabled devices. JXTA for J2ME is dependent upon HTTP which is the only protocol supported in the smallest profile, MIDP.

- **Security**: Both Jini Surrogate Architecture implementation and JXTA for J2ME rely on the security already present in the devices. In addition the relays are covered by the security mechanisms in the respective network technologies.

8.3 Interoperability between Jini and JXTA

Interoperability is the ability of software and hardware on multiple machines from multiple vendors to communicate. This can be achieved in different ways, as visualized in Figure 8-1.

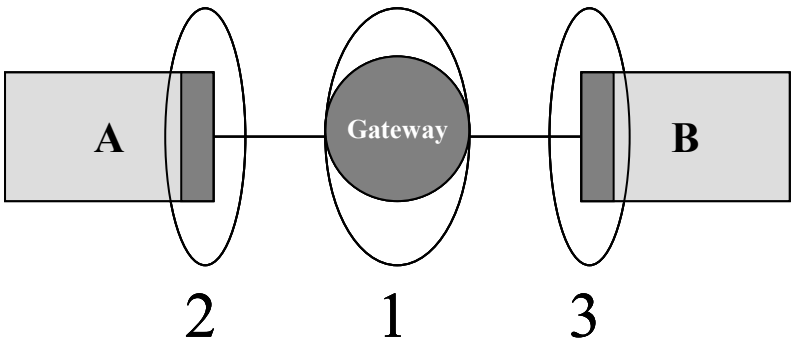


Figure 8-1 Interoperability between two technologies, A and B

Figure 8-1 shows a scenario where two technologies, A and B, want to communicate. There are three possible scenarios for how this can be achieved, as marked in the figure:

1. A gateway solves the differences and maps between the communication mode used in A and B.
2. A must adapt to support the way B communicate.
3. B must adapt to support the way A communicate.

The next section continues with a description of some sample solutions on how to achieve interoperability between Jini and JXTA, and then moves on to a discussion around the different solutions.

8.3.1 Solutions

Interoperability between Jini and JXTA is not as trivial as many think. The technologies are fundamentally different in that Jini requires Java, and JXTA is language independent, i.e. Jini is based on object serialization and proxies while JXTA peers communicate using XML. To be interoperable, some changes must be imposed in one way or another. It would be much easier if one considers Java as present in the network. All JXTA peer will then be able to communicate using both Java and XML. Things get much more complicated if Jini services should be added to a "pure" JXTA environment where services is implemented in different programming languages.

The different solutions described below are discussed in the next section:

- **(S - 1) Surrogate Brigde**
- **(S - 2) XML support in Jini**
- **(S - 3) JXTA as communication platform in Jini**
- **(S - 4) JXTA resources as Jini services**

(S - 1) Surrogate Brigde

The simplest solution is to use the surrogate architecture, and let the surrogate generate and parse XML documents. This solution will only create a bridge between Jini and JXTA, and the two networks will continue to be separate. JXTA resources can be announced on the Jini lookup service by the surrogate host on behalf of the JXTA resource, and JXTA peers can use Jini services by browsing the content in the Jini Lookup Service. The surrogate host themselves could be advertised as resources in the usual way on the JXTA network.

(S - 2) XML support in Jini Lookup Server

Another possibly solution to interoperability is to incorporate an XML parser in the Jini Lookup Service so that it is able to parse XML documents and assume that all JXTA resources are implemented in Java. The lookup service can then act as a JXTA Rendezvous peer, and parse JXTA Peer Advertisements and Peer Discovery. Peers can either use the JXTA solution of discovery through the Peer Discovery Protocol (PDP), or unicast discovery messages to a Jini Lookup Service. The services will be returned as serialized Java objects, where encoding of the object state is left to the peer. IDs in Jini and JXTA must be matched, for example by letting Jini IDs be matched to the peer IDs associated with each service, allowing JXTA peers to use XML to advertise and discover the services. This can again be matched to the Jini service ID to allow a serialized Java Object to be returned to the peer.

There is already a project going on to integrate an XML parser in the Jini architecture; Jump [Jump]. Several other changes are also proposed with the objective to embrace P2P in Jini. A project based on the Jump project, JumpStart [JumpStart], has proposed to incorporate Jini in JXTA with the assumption that Java developers wants a tool that allows objects to be moved around. The project caused a large discussion in the JXTA community, and has not been accepted as a JXTA project at present time. In addition to the above mentioned changes, JumpStart will use Jumpers[1], in the form of pre-configured set of peer routers, to forward messages on behalf of the peers.

(S - 3) JXTA as communication platform in Jini

A third solution is to make Jini use JXTA instead of RMI to communicate between nodes in the network. Since both Jini and JXTA are protocol frameworks, where Jini solves "application" protocol problems, while JXTA solves "transfer" protocol problems, it should be possible to use JXTA to transport Jini queries and proxies. Jini is not dependent upon RMI, and may gain by using JXTA which is independent of programming language and platform. JXTA messages can transfere any information, including proxies and Jini queries, and can include information describing the information included in the message. This way, JXTA resources with Java support will be able to communicate with Jini resources, that is Jini services appear as JXTA peers.

It has also been claimed that is should be possible, though non-trivial, to write a Jini framework in C++ or another programming language. In such an environment a language independent protocol like JXTA is needed.

(S - 4) JXTA resources as Jini services

The fourth solution is a suggestion to make JXTA resources available for Jini clients. A Java version of the JXTA resource is registered in the lookup server, and is available for Jini clients. The proxy downloaded to the client upon service request communicate with the JXTA peer using the JXTA protocols.

1. Jumpers, developed by Jumper Network [JumpNet], is a dynamic gateway that separates public and private address spaces.

8.3.2 Evaluation of the suggested solutions

The former section gave an overview of some of the different solutions to interoperability between Jini and JXTA. This section will discuss the solution in realtion to advantages and limitations.

(S - 1) Surrogate Bridge

The surrogate bridge is a simple, and attainable solution as of today. A reference implementation of the surrogate architecture exists, so does an IPInterconnect implementation.

The advantage of the solution is that it creates no dependence on platform or programming language required in the JXTA environment, as long as the surrogate representing the JXTA resource is implemented in Java. Jini services require no change, neither does JXTA resources. The drawback is that the Surrogate host may be the bottleneck if many JXTA peers tries to connect to Jini services. It also creates a dependency on a central resource, the surrogate host.

(S - 2) XML support in Jini

The XML support in Jini solution tighten the two network technologies in higher degree than the previous suggested solution. The advantages is that rather than just exchanging a document, peers will be able to exchange objects between mobile agents that allow them to do more than read the document they exchange but collaborat on them. Disadvantages is that several changes has to be made to the Jini lookup server, and some to the JXTA architecture. In addition the solution is dependent upon Java, which limits the peers that can participate in the network. It also creates problems like how the lookup server will know that an advertisment or discovery messages comes from a Java-enabled peer or not.

(S - 3) JXTA as communication platform in Jini

Using JXTA as communication platform will make Jini independent of RMI, which is the bottleneck for today's embedded, Java-enabled devices. This way they may be able to participate in the Jini community without use of the surrogate architecture. When it comes to the advantages in relation to JXTA, the solution will make more services available for Java-enabled JXTA peers. JXTA peers can get information about the Jini services, and use them if they have Java-support.

The disadvantage is that the solution requires several changes to the Jini implementation as well as in the lookup service. Lookup services and other services that wants to join the Jini network must be able to generate and parse XML, which is very resource consuming. But services might use binary messages, and in this way limit the burden. Another disadvantage is that not all JXTA peers can use the extended service area.

(S - 4) JXTA resources as Jini services

The S - 4 solutions is somewhat similar to the S - 3 solution, but concentrates on the JXTA resources. Advantages are that Jini clients will be able to use services both on the Jini network and on the JXTA network. But the solution impose several requirements on the peers, which should be spared from details in the interconnect. In addition only JXTA resources with Java capabilities will be able to use the solution.

8.4 Relation to other distributed network technologies

It might be interesting to look at Jini and JXTA in relation to two related distributed network technologies; Web Services and Microsoft .NET. This section first considers Web Services in relation to the technologies in question, and then takes a look on Microsoft .NET.

8.4.1 Web Services

Web Services consist of different technologies and models, and can be compared to other technologies in a multitude of ways. This discussion will concentrate on the Web Service Platform and the primary

Web Service Model: the Service Oriented Architecture (SOA). Technologies in the Web Service Platform are UDDI, SOAP and WSDL. UDDI is a centralized business registry that enables business-to-business communication, SOAP is designed to enable accessing remote service with other SOAP applications and WSDL is used to desribe the available services.

Jini

The objective of Jini and Web Services is somewhat similar, but target different areas. Where Jini is based on service delivery over a network, Web Services provides service delivery over the World Wide Web.

Web Service's SOA model share many similarities with the Jini architecture. Both depend upon a central server to keep track on the services available, and clients in the network get information about the services by asking the central server. Services publish their presence to the central server. The difference emerge upon closer examination of the technologies that constitutes the platform of both technologies. Where Web Services uses WSDL to describe the services, stores them in UDDI, and transfer the descriptions using XML and SOAP, Jini uses mobile code (proxies) to represent the services both in the central server and on the client requesting the service, and supports communication with the central server using RMI.

Jini could be used to develop Web Services, but is able to solve much more difficult issues such as scaling, change and complexity which are involved in state-of-the-art networking.

JXTA

JXTA and Web Services differs in a significant way when it comes to the architecture. While Web Services uses a central server to keep track of services, JXTA services are distributed in the network and announced using advertisements.

UDDI provides a similar service as JXTA, and both are based upon XML. Differences are that clients must register as UDDI clients to be able to access the system for searching and sharing of business opportunities, in contrast to JXTA where there is no sign-up process or fee. This way, JXTA is more loosely organized than UDDI users. SOAP uses XML, as do JXTA, and both have protocol specifications to enable remote service access. However, JXTA provides additional protocols and services to implement a full P2P application. Because both use XML, SOAP compliant messages can be sent between JXTA peers through JXTA pipes in a scenario where Web Services appear as JXTA resources.

8.4.2 Microsoft .NET

Microsoft .NET is a very diffuse platform, and the architecture might seem like a bunch of technologies that are grouped togheter to provide desired functionality: programming language independence and distributed computation. The best way to compare the technology is by looking on the intentions and how they are realized.

The objective of .NET is to provide interoperability among nodes in a network, and this is accomplished by providing a framework where applications can be written in any programming language (as long as it is supported in the runtime), and all applications are available if they uses the .NET framework.

Jini

Microsoft .NET uses concepts from the Java programming language, such as a common run-time execution environment. Jini and .NET have the same objective, to provide interoperability among nodes in a network, but the objective is accomplished in different ways. But while Jini uses Java and runs on many platforms, .NET is language independent. Microsoft also claims that .NET will be available on different platforms, but until now only Windows is supported. Jini depends upon Java, and as a result the services are independent on platform.

JXTA

As .NET, JXTA is independent of programming language, but in such a way that it also is platform independent. JXTA applications can communicate with other applications written in another language and execute on another platform than windows, i.e. it does not require any plug-in to handle further platforms, as does .NET.

The Web Services aspects of the .NET platform uses XML, but this does not alone make JXTA and .NET comparable since the two platforms have different goals. .NET focuses more on the traditional client/server architecture of service delivery, and might form the foundation of a P2P application, which would require developers to specify core P2P interaction such as peer discovery. These mechanisms are already defined by the JXTA protocols.

8.5 The Prototype

The prototype developed as part of the assignment demonstrate the use of J2ME MIDP and JXTA for J2ME in an instant messaging (chatting) application where the users could participate in different groups according to location.

The objective of the prototype was to explore possible ways to implement applications on mobile terminals, and get to know J2ME MIDP and JXTA for J2ME. One of the main conditions for an application development platform and related packages to succeed it to make them easy to learn and use. Both MIDP and JXTA for J2ME is relatively easy to use if a developer is familiar with the Java programming language. The main challenge is the limited resources, like cpu and memory, on most mobile devices. JXTA for J2ME imposed another challenge in that the APIs is under development. As a result the API can suddenly change, and so it did halfway through the development causing some rewriting of code.

During the development, both advantages and limitations of MIDP and JXTA for J2ME were revealed, in addition to several challenges related to design decisions and non-functional requirements. The next section will cover some main areas and discuss these in relation to JXTA for J2ME. This includes the design decisions, the non-functional requirements and possible other network technologies that could have been used. Design decisions have impact on JXTA for J2ME, and the non-functional requirements are a way to evaluate this network technology package.

8.5.1 Design Decisions

According to the object-oriented philosophy of Java, an application should be divided in object that can be reused when necessary. But as a result of the resource limitations of small devices, the developers may be forced to move away from this philosophy to be able to reduce the MIDlet memory usage. Objects must be allocated from runtime memory, as opposed to primitive data types (int, long, boolean, etc.), which are allocated directly on the stack. Even though the MIDP API consists of several classes, objects are only used when it clearly makes functional sense to use them. Otherwise, primitive data types are used.

Viewed in this light, the ITSSystem was designed with a relatively small number of classes. All code could have been placed in one single class, but the first non-functional requirement required network independence. In addition the development of the classes was to be shared between two developers. To make the application easier to understand, ITSClient and NetworkClient could have been divided into smaller classes. Since ITSSystem is a small program this would not have caused any problems, but a solution with minimal number of classes was chosen in order to optimize the code and make the implementation size as minimal and efficient as possible.

8.5.2 Non-functional requirements

The first two non-functional requirements, stating that the system should be network technology independent and run in SDK and JWT are fulfiled in ITSSystem 1.0. Non-functional requirements related to JXTA for J2ME are discussed in this section, while the rest can be found in appendix C.

Execution on a J2ME compliant device

The third non-functional requirement states that the prototype should run on a J2ME MIDP compliant device. Since access to a device with J2ME MIDP was limited, and the available iPaq was not J2ME compatible, a lot of time were used to find out how to execute the application on the iPaq. The tools and technology inspected during this phase are presented in appendix E, and the possible reasons for why we failed to test the application on the iPaq are presented here.

The Jeode platform seemed like a good solution since the emulating software had been tested by the JXTA for J2ME development team. The ITSSystem started on Jeode, but when the application tried to connect to the relay, the application closed, and no network connection seemed to be possible. As a result, the Jeode solution was turned down. A drawback is that the Jeode runtime environment with libraries is large, and only a small fragment is used by the emulator. In addition the emulator imposed certain problems due to the emulator being under development.

Next solution was the IBM Websphere. The solution took less space than Jeode, but needed the application packaged as an IBM file-type jxe (Java Executable) to make it run. As a result, the Websphere environment had to be set up on a computer to generate the file. This was a time-consuming process because of the insuffienct information and messy presentation of the information that was provided. The installation of the iPaq was neither a trivial task, but at the end it succeeded with help from other developers. Even so the application did not execute, but left an error message complaining that something was wrong with the jxe file.

Availability

NF - 6 states that a user should be able to connect to the network and register with the service in 90% of the cases. Network connection is dependent upon the device in use, and available network-connection capabilities for this device. The most likely solution for devices using the first version of the ITSSystem is using the GSM-network as access network to the Internet. Then the availability of the network is dependent upon the availability of GSM covered areas, and whether the base stations are busy or not, i.e. maximum other incoming or outgoing connection is reached. In addition the current application is dependent on the rendezvous peer that act as relay to be up and running.

Reliability

The non-functional requirement NF - 7, which is concerned about the reliability, has not been taking into consideration when implementing ITSSystem. The services of the application are inaccessible when a network is not reachable. But as long as the mobile terminal has contact with a network, there is nothing indicating that it cannot run undisturbed for 12 hours. If a user disconnects from the network, he cannot continue his session when logging on within 5 minutes since the state of the user is not stored anywhere. When the network connection closes, the user will go off-line, but he should still be able to execute the tasks that are not dependent on network connection, like editing the group list, etc.

Response time

NF - 9 states that the user should receive response from the relay within the set poll interval. As of today the messages are placed in a queue at the relay server, and the first message in the queue is returned upon request form a client. As a result, the desired response time is rather optimistic since it assumes that the next message in the queue is the response to the last request from the client. In situations with low traffic, this might be the case, but at times where several request are made between the poll intervals, for example

at start-up, there will be problems to fulfil the requirement. Another problem is when several messages are received from the network for example while performing a search. An improvement could be that the server could return all messages in the queue upon a request from the client.

8.5.3 Alternative network technology solutions

Another technology that could be used to connect to a network instead of JXTA is the Jini network technology. Jini could be used as a replacement to JXTA or in addition to JXTA. For example, if connected to the Internet, the mobile device could consult a lookup server (Jini Lookup Server) to find the groups available and avoid the time consuming job to search the JXTA network. On the other hand, if a connection to the Internet is out of range, JXTA mechanisms could be used to search for groups. This solution will create a demand for interconnect between the two technologies, and different scenarios were discussed in section 8.3. The most suitable would be the Surrogate Bridge since it require less work than the others.

The prototype could also have been implemented using pure http, and used a relay server in the form of a servlet, php or cgi script, to connect to distributed technologies, like CORBA or RMI (see Figure 3-11). Alternatively a client/server model could be used, where a central server could be implemented to host the service, and the mobile device could connect to the server, using HTTP. Such a solution would require the service to keep track of the users in a database, and store, forward and route messages to the right users. This would lead to much load on the server, which could be a bottleneck for the service. In addition the application would then be dependent on the infrastructure.

8.6 Jini and JXTA in AMIGOS

AMIGOS, Advanced Multimedia In Group Organized Services, is a service platform that enable users to establish and participate in Meeting Places, i.e. virtual rooms where interaction optionally take place between the participants. An interesting scenario would be to include Jini and/or JXTA in this service platform, and this section will discuss possibly scenarios for incorporation.

The below suggested solution for usage of Jini and JXTA impose a fundamental change to the suggested architecture for the AMIGOS service. They move it from being very centralized to becoming decentralized. Today's vision of AMIGOS is to use central places where the users can fetch information about meeting places, and where services are registered. In addition, visible meeting places for a user is dependent upon localization and user profile. Localization is handled by the central servers, and all the central servers share a repository that holds the user profiles.

In a distributed architecture for the AMIGOS service, the following components must be present:

- **User profiles that authenticate the users, and hold their preferences.**
- **Meeting Places that users can create, join, participate and leave.**
- **A Location server** .

The rest of this section will cover Jini solutions and JXTA solutions in relation to AMIGOS. In each of these two cases the suggested solutions are explained, then the advantages and limitations of the solutions are discussed.

8.6.1 Jini

Jini requires the use of the Java programming language, and therefore limits the availability of meeting places to a Java environment. Four possible scenarios are suggested:

- **A - Discovery of the AMIGOS server**
- **B - Decentralized responsibility**
- **C - The AMIGOS service as a JavaSpace**
- **D - Meeting Places as JavaSpaces**

A - Discovery of the AMIGOS server

The easiest solution is to use Jini to locate AMIGOS servers through the lookup service. That is, the AMIGOS server is represented as a Jini services and announced to the Jini lookup service. Services or clients that want to find the AMIGOS server send a request to the lookup server, and receives a proxy that handles all communication to the server. Authentication and the handling of meeting places and location is left to the AMIGOS server. Location servers can also be services themselves, allowing other Jini clients to find their position.

This solution will also allow discovery of peripherals, like printers or other useful entities in the network.

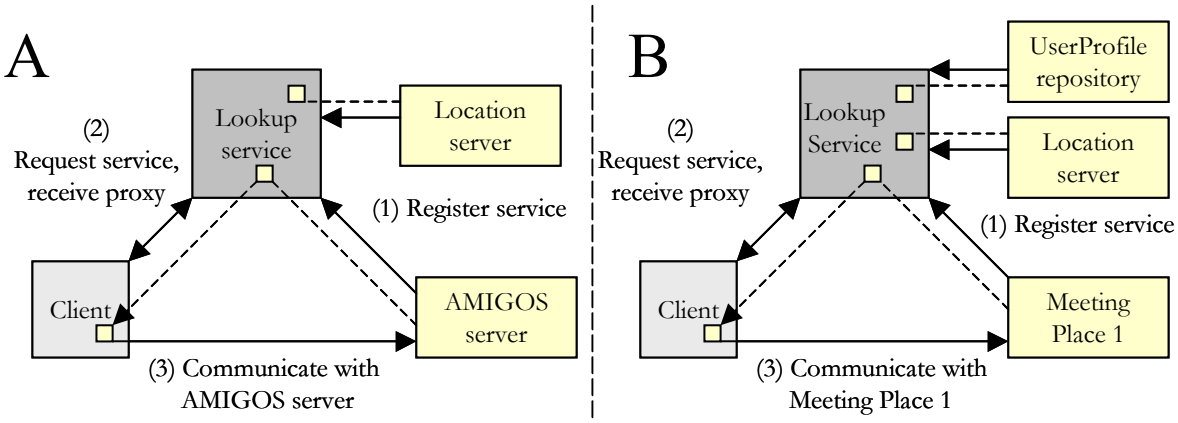


Figure 8-2 Jini in AMIGOS, solution A and B

B - Decentralized responsibility

A more advanced solution is to split the responsibility of the AMIGOS server in several parts that in cooperation constitutes and fulfil all services in AMIGOS. Meeting places and location servers appear as Jini services, and the authentication is left to the meeting places. The central user profile repository can also appear as a Jini service, and be used by the meeting places to authenticate the users that wants to join the meeting place. This solution provide a highly distributed environment, with the advantages and limitations that implies. The solution also moves the Meeting Places from being virtually and globally available, to being localized on a unit in the Jini environment.

C - The AMIGOS service as a JavaSpace

Another Jini solution is to use JavaSpace Service to provide meeting places. JavaSpace is a Jini service that provides a distributed, persistence mechanism for objects in the Jini community. A JavaSpace can be found through Jini Lookup Services, and can for example hold meeting places for a certain location or area. As Jini, JavaSpace uses leasing that ensures if a service leaves the room the leases will expire and the connected users will be disconnected from the service. This solution have similarities to the first solution, only that the AMIGOS server is replaced with a JavaSpace. Users can create meeting places and

place them in the JavaSpace for others to find. Solution C solves the complexity introduced in solution B: when a JavaSpaces is located, all available Meeting Places can be browsed.

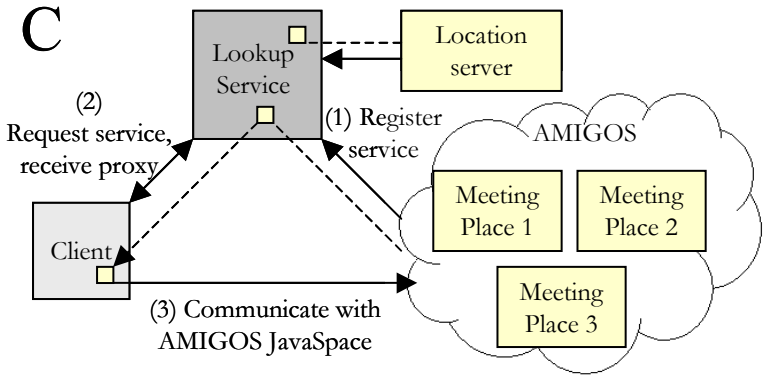


Figure 8-3 Jini in AMIGOS, solution C

D - Meeting Places as JavaSpaces

Representing Meeting Places as JavaSpace have similarities with solution C. The difference is that a JavaSpace is one Meeting Place, and not the total AMIGOS service. This leaves more to up Jini mechanisms. The Meeting Places can be found through the lookup service using templates setting requirements for the search, for example location or interests. This requires the user to know his user profile, and possibly his location.

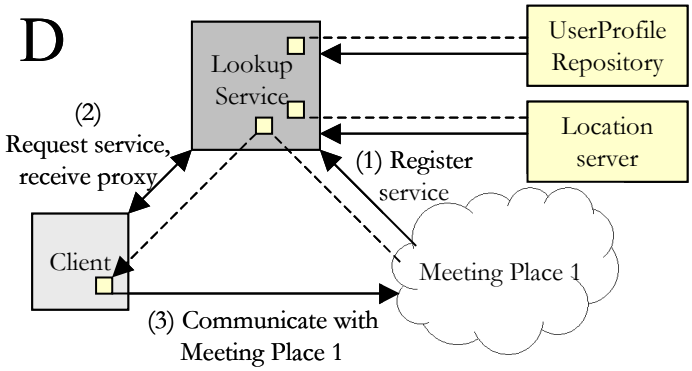


Figure 8-4 Jini in AMIGOS, solution D

Advantages and limitations

Solution A only provides a discovery mechanism, and as a result the clients are not dependent on knowing the exact location of the AMIGOS server. The advantage is that a proxy represents the AMIGOS server in the Jini community and makes the client independent when it comes to communication protocol, i.e. the client does not need to know how to communicate with the AMIGOS server since this communication is handled by the proxy. The lease mechansim of Jini also ensurs fault tolerance, that is if the AMIGOS server goes down the clients will not be able to renew the lease and as a result assume that the service is unavailable.

The second solution, solution B, makes it easy to for a user to create meeting places and make it available for others with no intervention from other parties. Creators of meeting places are responsible for the management, and can withdraw the meeting place from the lookup service at any time without causing

faults as a result of the lease mechanism. The initiator also sets up the requirements in connection to the meeting place, like who is allowed to participate, and the meeting place is responsible for authenticating the user. Since lookup services will only have overlapping information about meeting places, and a user connects to the nearest meeting place, it might be possible to assume that all registered meeting places in a lookup service are in the vicinity of the lookup service. Meeting places can be found according to location and user profile, which require the user to know his position and user profile. The search can be based on templates that states requirements for the meeting place, for example type of interest, location and so on. Browsing meeting places is possible based on the same mechanism.

Drawbacks of solution B are that meeting places must handle the authentication of the user, and that there are no mechanism to authenticate the user before he gets a list of existing meeting places if this mechanism is not built into the client application itself. This could be solved by forcing authorization after discovery. Solution B also introduces another complexity: a client cannot have the total overview of all meeting places. It has to initiate a search that does not guarantee that all the existing meeting places are found.

Solution C suggests the use of JavaSpace. The advantage is a persistent environment where meeting places can exist. It relies on the mechanisms in JavaSpace, among other it leaves the authentication to the JavaSpace. This solution has similarities to solution A, and share the same advantages and limitations.

Solution D suggest that JavaSpaces can themselves be meeting places. As solution C it relies on the mechanisms of JavaSpace, but authentication of the user is left to the Jini environment. The Jini environment has no mechansims to provide such authentication before a user contacts a lookup service. The solution has similarities to the decentralized solution (B), and share the same advantages and disadvantages.

8.6.2 JXTA

As Jini, JXTA does not consider the implementation of the meeting places and concentrates on how to find the different entities in the AMIGOS environment. The only requirement is that all entities must be able to generate and parse XML. There are three different scenarios of using JXTA in AMIGOS:

- **A - Discovery of the AMIGOS server**
- **B - Decentralized responsibility**
- **C - Meeting places as peer groups**

A - Discovery of the AMIGOS server

The first scenario suggest that the AMIGOS server appear as a JXTA peer. It advertise its presence and clients find the peer, or resource, by using the JXTA discovery protocol. Authentication and handling of meeting places and location is left to the AMIGOS server.

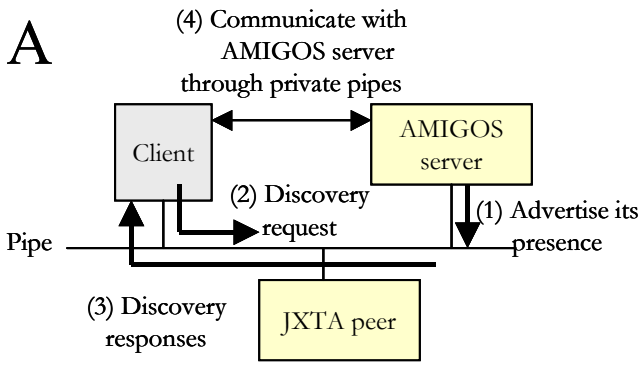


Figure 8-5 JXTA in AMIGOS, solution A

This solution will also allow discovery of peripherals, like printers or other useful entities in the network.

B - Decentralized responsibility

A decentralized responsibility aims at splitting the responsibility of the AMIGOS server in several parts which in cooperation constitutes and fulfils the total of servces in AMIGOS. Meeting places, location servers and devices appear as JXTA peers, and are members of the same peer group. The peer group is responsible for authenticating the peers joining the group, and only members of the group are able to see the other members.

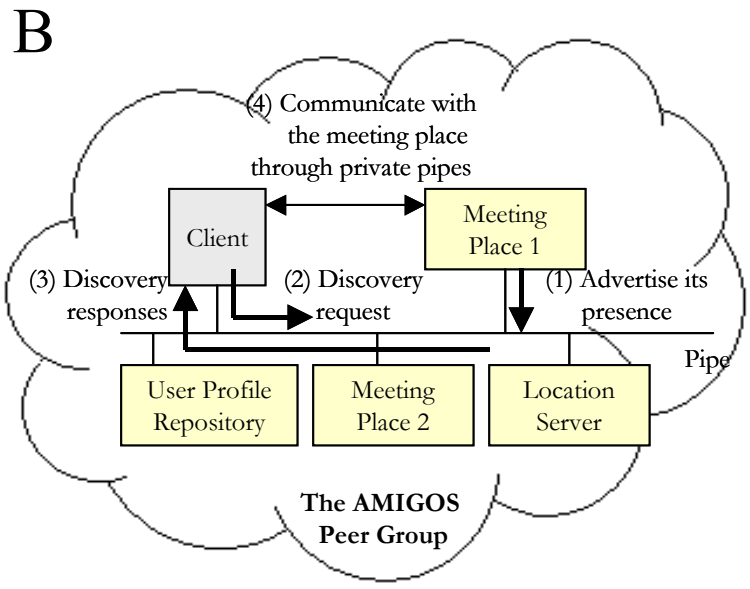


Figure 8-6 JXTA in AMIGOS, solution B

Meeting places can be created, and then join the AMIGOS peer group. They announce their presence through advertisements to the peer group. The advertisement holds information about the meeting place like location, requirements for joining and so on. To discover a meeting place or browse meeting places, a peer first need to join the peer group. Then the peer are able to send a discovery message to the peer group. The peer that created and announced a specific meeting place must manage it, like setting requirements that limits the users that can join the meeting place, and giving users access.

C - Meeting places as peer groups

The last suggested solution is based on the previous solution explained above, but differs in one significant way. Meeting Places are peer groups themselves, and it is the joint responsiblity for the peers in a peer group to manage the it, like setting requirement for access and giving access.

Advantages and limitations

The first suggested JXTA solution only introduces discovery of the AMIGOS server, and as a result remove the dependency on pre knowledge in the clients when it comes to the location of the AMIGOS server. In addition it introduces JXTA mechanisms like fault tolerance. As communication is done using XML, this allows for the entities to be implemented in any programming language and run on any plat-

form. The solution also introduces mechanisms to further support usage of other resorces in the network like printers, scanners, etc.

Solution B introduces a decentralized mechanism that is more fault tolerant than the previous solution since the responsibility is shared among many entities in the network. The solution solved the problem with authorization since a peer must be authorized before it is allowed to join the AMIGOS group and browse the Meeting Places. But it also introduces a complexity: a client cannot have the total overview of all meeting places. It has to initiate a search for meeting places that does not guarantee that all existing meeting places are found.

The last suggested solution uses nested groups, a functionality that is not dictated in the specification. As of today the mechanism is not available in the reference implementation, but it introduces new possibilities as sketched in solution C. The main group represents the AMIGOS service, and the peers in the peer group handle authentication and jointly have overview of the meeting place peer groups. The distributed environment can be both an advantages and a disadvantage since all assignments are shared among the participating parties. As solution B, solution C introduces a complexity in relation to the total overview of meetins places: a peer must initiate a search for meeting places that does not guarantee that all existing meetin places are found.

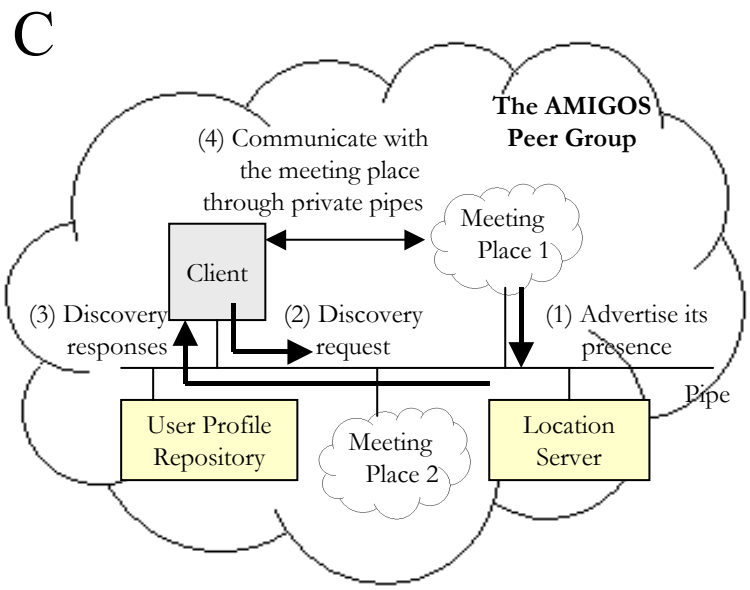


Figure 8-7 JXTA in AMIGOS, solution C

8.7 Scenarios for the future

There are different scenarios on how the future of distributed peer-to-peer networks will be, and which solution is the best. This section will point on some main areas of interests.

8.7.1 Personal Area Network (PAN)

Jini and JXTA is the first step towards an environment where all devices can discover each other and form Personal Area Networks (PAN), that is to say network technologies are independent of existing or non-existing infrastructure. Using JXTA on top of an ad-hoc network technology that is based on a radio interface will provide such an environment. The goal of JXME is this PAN scenario, in addition to independence from the relay, which requires more resources than there exists today.

8.7.2 Improvements to JXTA

Discovery in Jini and JXTA takes time today, but improvments to the technology and transport network technologies might improve this. If JXTA is to succeed it has to prove the community that it is a framework with advantages more promisting than that of Microsoft .NET and other competitors. In the end, the success of JXTA lies in its ability to move from a research project to a useable framework that is adopted by developers and the community. Other improvements to JXTA is to define what type of applications the technology is best suited for. Today this is left to the community to decide on which applications will succeed and which will fail. As Li Gong said in [Comp2001]: "JXTA's version 1.0 is more a starting point for the developer community than a finished product."

8.7.3 The existence of an optimal solution

Distributed network technologies will continue to be discussed and compared, argumenting that one of them is THE solution. But none are THE solution to every problem, they are all just tools and all have their place. The challenge is to choose the one that make sense for the problem at hand.

Chapter 9

Conclusion

"My interest is in the future because I am going to spend the rest of my life there."
Charles F. Kettering

Embedded devices has gained great success the last couple of years, and their real value will become visible when network capabilities are introduced. As these devices have limited capabilities and are highly mobile, the network environment are required to handle these challenges. The most suitable network architecture is a distributed one that follows the peer-to-peer paradigm, since these architectures can handle that clients come and go with no further notice. Peer-to-peer architectures are concerned with independent of infrastructure at the application layer, and handle everything from discovery of resources to usage of these resources without pre knowledge of their localization.

The solution for embedded devices, and the hope for the future, is that they are able to form private networks, Personal Area Networks (PANs), when they come in reach of each other. Peer-to-peer provides the necessary functionality at the application layer, while ad-hoc technologies like WLAN and Bluetooth handles routing and data transmission at the lowest layer in the protocol stack. A PAN will create exiting possibilities when it comes to sharing of information and usage of the available resources in the network.

Network solutions for embedded devices are dependent upon the capabilities in the programming environment on the devices. J2ME MIDP, which is a Java programming environment for mobile phones, provides simple protocol support that limits the network capabilities. But use of a middleman, or relay, makes the Java-enabled devices able to join in networked applications. The future promise that devices will be able to operate in PANs, and the next version of MIDP will introduce further network support that will enhance the networking capabilities.

Several years will pass before the enhanced specification of MIDP is available in devices. MIDP 1.0 was released in september 2001 and is just starting to be implemented in mobile phones. The first phone with Java-support came in fall of 2001, and Nokia released their first in May 2002. A number of other phones are announced to come during the year, but as people buy fewer phones it will take time to build a base of users. MIDP is more likely to gain success with the introduction of GPRS and 3G if its capabilities is made visible for the potential users. Adding new features requires more resources on the devices, which is likely to increase according to Moore's law.

The combination of MIDP and the Jini surrogate architecture creates possibilities for the device to utilize resources that otherwise would have been unavailable, for example printers, scanners and other computers. At the moment Jini is not recommended to be used in commercial applications since the surrogate architecture is young, not fully tested and needs further research. Requests to the lookup service has also proven to be slow, and the delay will limit the usability of applications since users require fast responses. Even so, the Jini surrogate architecture seems more thought out than the JXTA alternative when it comes to giving all devices access to a networking environment, Java-enabled or not.

JXTA is very promising with its independence on both programming language and platform. What might prevent the use of the architecture is the complexity, i.e. many concepts and protocols to keep track of. In my opinion, this is the most suitable architecture for highly mobile, Java-enabled devices in the future since the JXTA for J2ME package is easy to use and does not necessarily require any knowledge of JXTA. The conditions are that J2ME will gain success and that the capabilities of MIDP increase. Since the major mobile terminal manufacturers have announced J2ME MIDP support in their phones, and take part in the standardization, I believe that these conditions will be fulfilled within the next few years.

When it comes to developing commercial applications using JXTA, I will not recommend this at the current time. JXTA for J2ME relies on the reference implementation, which has been reported to be unstable, unreliable and slow. In addition, JXTA for J2ME is under constant change, and does not support all the advanced features that would give applications value, like full search capabilities. A commercial application may be possible some time during 2003 since the JXTA reference implementation constantly changes and is improved by the JXTA community.

Jini and JXTA have several possible areas of usage in AMIGOS, which is a service platform that enable users to establish and participate in virtual rooms called Meeting Places. An introduction to these distributed architectures will bring the AMIGOS architecture from being very centralized to being distributed, with all the challenges this introduces. Some examples are reduced consistency of data, reduced security and loss of central control. Possible scenarios range from just discovering AMIGOS servers to spliting the AMIGOS service in several parts that co-operate to provide the full service. In my opinion, JXTA is the most suitable solution since it has more features that cover the needs of AMIGOS, than Jini. A problem with Jini could be authentication of the user since Jini has no mechanism to support this. The platform and programming independence in JXTA will make AMIGOS available for all devices. JXTA also has the mechanisms to divide the environment in several smaller environments , peer groups, which will suite the Meeting Place scenario presented in AMIGOS.

Today, devices that form ad-hoc network and communicate peer-to-peer is just an area of research. Devices have the potential, but neither ad-hoc and peer-to-peer technologies nor programming environment for embedded devices are fully developed. In addition the smallest one lack the necessary resources to be able to participate. Embedded devices need a "killer application" to make people see the potential and usage area of them, and until the time where all these factors are present we must be content with what is available.

Chapter 10

Abbreviations

AMIGOS	Advanced Multimedia In Group Organised Services
API	Application Programming Interface
AVANTEL	Advanced Telecom Services
CA	Certificate Authority
CDC	Connected Device Configuration
CLDC	Connected, Limited Device Configuration
CLR	Common Language Runtime
CTS	Common Type System
DES	Data Encryption Standard
DNS	Domain Name Server
DSA	Digital Signature Algorithm
ERP	Endpoint Routing Protocol
FTP	File Transfer Protocol
GAP	Generic Access Profile
GOEP	Generic Object Exchange Profile
GPRS	General Packet Radion Service
GSM	Global System for Mobile Communication
HTTP	Hypertext Transfer Protocol
HTTPS	Secure HTTP
IEEE	Institute of Electrical and Electronics Engineers, Inc.
IEFT	Internet Engineering Task Force
IMPS	Instant Messaging and Presence Service
IP	Internet Protocol

ITSSystem	Intelligent Traffic Service System
J2EE	Java 2 Platform, Enterprise Edition
J2ME	Java 2 Platform, Micro Edition
J2SE	Java 2 Platform, Standard Edition
JAD	Java Application Descriptor
JAR	Java Archive
JCA	Java Cryptography Architecture
JCE	Java Cryptography Extension
JCP	Java Community Process
JNI	Java Native Interface
JSR	Java Specification Request
JVM	Java Virtual Machine
JXME	JXTA for J2ME
JXTA	Juxtapose
KVM	Kilobyte/Kuaui Virtual Machine
L2CAP	Logical Link Control and Adaption Protocol
LAN	Local Area Network
MID	Mobile Information Device
MIDP	Mobile Information Device Profile
MMS	Multimedia Messaging Service
NAT	Network Address Translation
OBEX	Object Exchange Protocol
OTA	Over-The-Air
P2P	Peer-to-Peer
PAN	Personal Area Network
PATS	Program for Advanced Telecom Services
PBP	Pipe Binding Protocol
PDA	Personal Digital Assistant
PDAP	PDA Profile
PDP	Peer Discovery Protocol
PEP	Peer Endpoint Protocol
PIP	Peer Information Protocol

PKI	Public Key Infrastructure
PMP	Peer Membership Protocol
PRP	Peer Resolver Protocol
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RVP	Rendezvous Protocol
SDAP	Service Discovery Application Profile
SDP	Service Discovery Protocol
SHA-1	Secure Hash Algorithm
SIMPLE	SIP for Instant Messaging and Presence Leveraging Extensions
SIP	Session Initiation Protocol
SIPPING	Session Initiation Proposal Investigation
SMS	Short Message Service
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SPP	Serial Port Profile
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDDI	Universal Description, Discovery and Integration
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
VAME	Visual Age Micro Edition
WAP	Wireless Application Protocol
WLAN	Wireless LAN
WSDL	Web Service Description Language
XML	Extensible Markup Language

Chapter 11

Reference Library

[3GPPsip] *3GPP requirements on SIP* [online] http://www.ietf.org/internet-drafts/draft-garcia-sipping-3gpp-reqs-03.txt

[829-1991] *829-1991 IEEE Standard for Software Test Documentation from 1991* [online] http://www.ustreas.gov/hrsolutions/docs/test/

[Acc008] *Motorola Accompli 008 Features* [online] http://developers.motorola.com/developers/wireless/global/uk/A008.html

[Amigos] A*dvanced Multimedia In Group Organised Services (AMIGOS)* [online] http://www.item.ntnu.no/avantel/AMIGOS.html

[Amigos01] *Rational behind AMIGOS*; see [Amigos]

[Amigos02] *Dictionary for AMIGOS*; see [Amigos]

[Amigos03] *Requirements Specification for AMIGOS*; see [Amigos]

[Amigos04] *Use Cases for AMIGOS*; see [Amigos]

[Amigos05] *Examples of Meeting Places in AMIGOS*; see [Amigos]

[Anhinga] *The Anhinga Project* [online] http://www.cs.rit.edu/~anhinga/

[Avantel] *Advanced Telecom services (AVANTEL)* [online] http://www.item.ntnu.no/avantel/

[Bouncy] *Bouncy Castle cryptography package* [online] http://www.bouncycastle.org/

[Brandis] *Joakim von Brandis,* Summer job at PATS with focus on location services in AMIGOS, June 2002

[CLDC1.1] *JSR-139 Connected Limited Device Configuratin 1.1* [online] http://jcp.org/jsr/detail/139.jsp

[Comp2001] Gong, Li; *JXTA: A Network Programming Environment*; IEEE Internet Computing, May-June 2001.

[Comp2001] Gong, Li; Sun Microsystems Inc.; *JXTA: A Network Programming Environment;* IEEE Internet Computing, volume 5, Issue 3, May-June 2001 , page 88-95

[Computer] *Computer*; Volum 34, Issue 7, July 2001, page 21

[Computing] Golden G. Richard III; *Service Advertisement and Discovery: Enabling Universal Device Cooperation ;* IEEE Internet Computing, September-October 2000.

[EJava] *EmbeddedJava Application Environment* [online] http://java.sun.com/products/embeddedjava/

[Eric2000] Frodigh, Magnus; Johansson, Per; Larsson, Peter; *Wireless ad hoc networking - The art of networking without a network* ; Ericsson Review No. 4 2000[online] http://www.ericsson.com/about/publications/review/2000_04/article124.shtml

[Forte] *Forte for Java* [online] http://wwws.sun.com/software/Developer-products/ffj/

[Frisk] Frisk, Charlotte; Broberg, Ronny; *Web Services in Telenor*; Master thesis at Department of Telematics, NTNU; June 2002

[GNUpl] *GNU public license* [online] http://www.fsf.org/copyleft/gpl.html

[Gnutella] *Gnutella* [online] http://www.gnutella.com/

[Gollmann] Gollmann, Dieter; *Computer Security*; John Wiley & sons, 1999

[Gong] Gong, Li; *Project JXTA: A Technology Overview*; [online] http://www.jxta.org/project/www/docs/TechOverview.pdf

[Graham] Graham, Ross Lee; Intelligent Information Systems LAB; Linköpings university; [online]http://www.ida.liu.se/conferences/p2p/p2p2001/hybrid.html

[Gupta] Gupta, Vipul; *Bringing Big Security to Small Devices*; 2001 JavaOne presentation; http://playground.sun.com/~vgupta/KSSL/2246gupta.pdf

[IBM] *IBM's WebSphere* [online] http://www.embedded.oti.com/wdd/

[IBMrt] *IBM's WebSphere, platforms supported* [online] http://www.embedded.oti.com/compat/

[IEEE] *Institute of Electrical and Electronics Engineers, Inc*. [online] http://www.ieee.org/

[Insignia] *Insignia* [online] http://www.insignia.com

[iPaq] *Compaq iPaq* [online] http://www.compaq.com/products/handhelds/index.html

[IPInt] *IP Surrogate Project* [online] http://developer.jini.org/exchange/projects/surrogate/IP/index.html

[IPIntSpec] *Jini Technology IP Interconnect Specification* [online] http://developer.jini.org/exchange/projects/surrogate/IP/sa-ip.pdf

[ITPro] Stang, Mark; Whinston, Stephen; *Enterprise Computing with Jini Technology*; IT Pro January-February 2001

[ITProf] Yeager, W; Williams, J; *Secure Peer-to-Peer networking: the JXTA example*; IT Professional, Volume 4, Issue 2, Mars/April 2002, page 53-57

[J2ME_spec] *Java Specifications Requests related to J2ME* [online]
http://jcp.org/jsr/tech/j2me.jsp

[J2MEFAQ] *J2ME Frequently asked questions* [online]
http://java.sun.com/j2me/faq.html

[J2MEwtk] *J2ME Wireless Toolkit* [online]
http://java.sun.com/products/j2mewtoolkit/

[JAIN] *The JAIN APIs* [online]
http://java.sun.com/products/jain/

[JavaJXTA] *Platform project, the Java 2 Standard Edition (J2SE) version of JXTA* [online]
http://platform.jxta.org/

[JavaSpace] *JavaSpaces Service Specification* [online]
http://wwws.sun.com/software/jini/specs/jini1.1html/js-title.html

[JCard] *Java Card Technology* [online]
http://java.sun.com/products/javacard/

[JCP] *Java Community Process* [online]
http://jcp.org/

[Jini] *Jini Network Technology - Sun's official Jini site* [online]
http://www.sun.com/jini

[JiniJO02] *Jini@ JavaOne Webcast*; March 25-29, 2002; San Francisco [online]
http://www.jini.org/content/webcast/Apr2002/index.html

[JiniOrg] *Jini.org* - the Jini community's web site[online]
http://jini.org

[JiniSA] *Jini Surrogate Architecture Specification* [online]
http://developer.jini.org/exchange/projects/surrogate/

[JiniSAO] *The Jini Technology Surrogate Architecture Overview* [online]
http://surrogate.jini.org/overview.pdf

[JiniSASpec] *Jini Technology Surrogate Architecture Specification* [online]
http://www.jini.org/standards/sa.pdf

[JiniWD] *Jini WirelessDevice Project* [online]
http://developer.jini.org/exchange/projects/wirelessdevice/

[JLCA] *Java Language Conversion Assistant* [online]
http://msdn.microsoft.com/vstudio/downloads/jlca/default.asp

[JSR 179] *Location API for J2ME* [online]
http://jcp.org/jsr/detail/179.jsp

[JSR-113] *Java Speech API 2.0* [online]
http://jcp.org/jsr/detail/113.jsp

[JSR-120] *JSR 120 Wireless Messaging API* [online]
http://jcp.org/jsr/detail/120.jsp

[JSR-129] *JSR-129 Personal Basis Profile Specification* [online http://jcp.org/jsr/detail/129.jsp

[JSR-134] JSR-134 Java Game Profile [online] http://jcp.org/jsr/detail/134.jsp

[JSR-135] *JSR 135 Mobile Media API* [online] http://jcp.org/jsr/detail/135.jsp

[JSR-164] *JSR 164 JAIN SIMPLE Presence* [online] http://jcp.org/jsr/detail/164.jsp

[JSR-165] *JSR 165 JAIN SIMPLE Instant Messaging* [online] http://jcp.org/jsr/detail/165.jsp

[JSR-172] *JSR 172 J2ME Web Service Specification* [online] http://jcp.org/jsr/detail/172.jsp

[JSR-177] *JSR 177 Security and Trust Services API for J2ME* [online] http://jcp.org/jsr/detail/177.jsp

[JSR-180] *JSR 180 SIP API for J2ME* [online] http://jcp.org/jsr/detail/180.jsp

[JSR-186] *JSR 186 JAIN Presence* [online] http://jcp.org/jsr/detail/186.jsp

[JSR-187] JSR 187 *JAIN Instant Messaging* [online] http://jcp.org/jsr/detail/187.jsp

[JSR-30] *JSR-30 J2ME Connected, Limited Device Configuration* [online] http://jcp.org/jsr/detail/30.jsp

[JSR-36] *JSR-36 J2ME Connected Device Configuration* [online] http://jcp.org/jsr/detail/36.jsp

[JSR-37] *JSR-37 Mobile Information Device Profile* [online] http.//jcp.org/jsr/detail/37.jsp

[JSR-46] *JSR 46 Foundation Profile* [online] http://jcp.org/jsr/detail/46.jsp

[JSR-62] *JSR-62 Personal Profile Specification* [online] http://jcp.org/jsr/detail/62.jsp

[JSR-66] *JSR-66 RMI Optional Package Specification Version 1.0* [online] http://jcp.org/jsr/detail/66.jsp

[JSR-75] *JSR-75 PDA Profile for the J2ME Platform* [online] http://jcp.org/jsr/detail/75.jsp

[JSR-82] *JSR 82 Java SPIs for Bluetooth* [online] http://jcp.org/jsr/detail/82.jsp

[Jump] *Project JumpStart*; [online] http://www.jumpernetworks.com/jump.html

[JumpNet] *Jumper Network*; owner of the Jump [Jump] and JumpStart [JumpStart] project; [online] http://www.jumpernetworks.com/

[JumpStart] *Project Jump*; [online] http://www.jumpernetworks.com/jumpstrt.html

[JXME] *JXTA fro J2ME Project page* [online] http://jxme.jxta.org/

[JXMEwp] Arora, Akhil; Haywood, Carl; Pabla, Kuldip Singh; *JXTA for J2ME - Extending the Reach of Wireless With JXTA Technology*; Sun Microsystems, Inc; March 2002 [online] http://www.jxta.org/project/www/docs/JXTA4J2ME.pdf

[JXTAdraft] Internet-Draft at IETF. JXTA v1.0 Protocols Specification - http://www.ietf.org/internet-drafts/draft-duigou-jxta-protocols-00.txt

[JXTAfaq] *JXTA Frequently Asked Questions* [online] http://www.jxta.org/project/www/docs/DomainFAQ.html

[JXTASec] Sun Microsystems, Inc; *Security and Project JXTA* [online] http://www.jxta.org/project/www/docs/SecurityJXTA.PDF

[JXTASpec] *JXTA v1.0 Protocols Specification* [online] http://spec.jxta.org/v1.0/docbook/JXTAProtocols.html

[JXTAvn] Traversat, Bernard; Abdelaziz, Mohamed; Duigou, Mike; Hugly, Jean-Christophe; Pouyoul, Eric; Yeager, Bill; *Project JXTA Virtual Network*; Sun Microsystems, Inc. ; February 2002 [online] http://www.jxta.org/project/www/docs/JXTAprotocols.pdf

[JXTAws] *JXTA web site* [online] http://jxta.org/

[Kiely] Kiely, Don; *Wanted: Programmers for Handheld Devices*; Computer, Volum 34, Issue 5, May 2001, page 12-14

[Knudsen] Knudsen, Jonathan; *Wireless Java: Developing with Java 2 Micro Edition*; June 2001

[Krishnan] Krishnan, Navaneeth; *The Jxta solution to P2P*; [online] http://www.javaworld.com/javaworld/jw-10-2001/jw-1019-jxta-p.html

[kSOAP] *The kSOAP project - SOAP for J2ME* [online] http://www.ksoap.org/

[KTopley] Topley, Kim, *J2ME in a Nutshell*; O'Reilly, March 2002

[Kumaran] Kumaran, S. Ilango; Jini Technology. An Overview ; Prentice Hall, Inc. 2002

[kXML] *The kXML project* [online] http://www.kxml.org

[Lyng2001] Lyngstad, Bjørg; *Java on Mobile Devices*; Project Assignment; Norwegian University of Science and Technology. Faculty of Electrical Engineering and Telecommunications. Department of Telematics (ITEM), Autumn 2001

[MahLor] Qusay H. Mahmoud, Nicholas Lorain; *Wireless Software Design Techniques* [online] http://wireless.java.sun.com/midp/articles/uidesign/

[Mahmoud] Mahmoud, Qusay ; "Wireless Java Security"; [online] http://wireless.java.sun.com/midp/articles/security/

[me4se] Micro Edition for Standard Edition (me4se) [online] http://me4se.org/

[MIDP1.0] *Mobile Information Device Profile 1.0* [online] http://java.sun.com/products/midp/

[MIDP2.0] *JSR 118 Mobile Information Device Profile 2.0* [online] http://jcp.org/jsr/detail/118.jsp

[Minar] Minar, Nelson; *Distributed Systems Topologies: Part 2*; [online] http://www.openp2p.com/pub/a/p2p/2002/01/08/p2p_topologies_pt2.html

[Napster] *Napster* [online] http://www.napster.com/

[NET] *Microsoft .NET site* [online] http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28000519

[NETcfaq] *Frequently Asked Questions About the Microsoft .NET Compact Framework* [online] http://msdn.microsoft.com/vstudio/device/compactfaq.asp

[NETcomp] *The .NET Compact Framwork Overview* [online] http://msdn.microsoft.com/vstudio/device/compactfx.asp

[NETfaq] *Microsoft .NET Frequently Asked Questions (FAQ)* [online] http://msdn.microsoft.com/library/default.asp?URL=/library/techart/faq111700.htm

[NETfto] *Microsoft .NET Framework Technical Overview* [online] http://www.gotdotnet.com/team/framework/DotNet%20Framework%20Technical%20Overview%20v3.doc

[NETJ#] *Microsoft Visual J# .NET Beta 2* [online] http://msdn.microsoft.com/visualj/jsharp/beta.asp

[NETJava] *.NET and Java* [online] http://www.gotdotnet.com/team/java/

[NETov] *.NET Framework Product Overview* [online] http://msdn.microsoft.com/netframework/prodinfo/overview.asp

[Pats] *Program for Advanced Telecom Services (PATS)* [online] http://www.item.ntnu.no/avantel/pats.html

[PJava] *PersonalJava* [online] http://java.sun.com/products/personaljava/

[Pnglib] *Png library from sixlegs.com* [online] http://www.sixlegs.com/software/png/

[PsiNaptic] *PsiNaptic* [online] http://www.psinaptic.com/

[PureTLS] *Claymore Systems* - the developer of Pure TLS [online] http://www.claymoresystems.com/

[Qusay] Mahmoud, Qusay; *Advanced MIDP Networking, Accessing Using Sockets and RMI from MIDP-enabled Devices*; January 2002 http://wireless.java.sun.com/midp/articles/socketRMI/

[RFC2396] *RFC2396 Uniform Resource Indicator (URI): Generic Syntax* [online] http://www.ietf.org/rfc/rfc2396.txt

[RFC2543] *SIP: Session Initiation Protocol* [online] http://www.ietf.org/internet-drafts/draft-ietf-sip-rfc2543bis-09.txt

[RFC2616] *RFC2616 Hypertext Transfer Protocol - HTTP 1.1* [online] http://www.w3.org/Protocols/rfc2616/rfc2616.html

[SETI] *SETI@Home* [online] http://setiathome.berkeley.edu/

[SFrame02] Bræk, Rolv; Husa, Knut Eilif; Melby, Geir; *ServiceFrame Whitepaper* (draft); Ericsson, May 2002.

[SIMPLE] *SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE)*; Includes 4 internet drafts [online] http://www.ietf.org/ids.by.wg/simple.html

[SIP] *Session Initiation Protocol (SIP)*; Includes 25 internet drafts [online] http://www.ietf.org/ids.by.wg/sip.html

[SIPPING] *Session Initiation Proposal Investigation (SIPPING)*; Includes 12 internet drafts [online] http://www.ietf.org/ids.by.wg/sipping.html

[SkaKau] Skaflestad, Odd Arild; Kaurel, Nina; *Peer-to-Peer Networking. Configuration Issues and Distributed Processing*; Arcticle written in realtion to subject SIE50AC Autumn 2001

[SOAP] *Simple Object Access protocol (SOAP) Protocol Specification v. 1.1* [online] http://www.w3.org/TR/SOAP

[Spotless] The Spotless System [online] http://research.sun.com/spotless/

[Sun] *Sun Microsystems, Inc.* [online] http://www.sun.com/

[SWaba] *Superwaba* [online] http://www.superwaba.org/

[telecom01] *Microsofts .NET i mobile terminaler*, Telecom, April 2002 [online] http://www.telecom.no/art/5064.html

[TLS] *Transport Layer Security (TLS)* [online] http://www.ietf.org/rfc/rfc2246.txt

[Topley] Topley, Kim; *Building Wireless Web Clients: Pitfalls of MIPD HTTP, Part 1* [online] http://www.onjava.com/pub/a/onjava/2002/04/17/j2me.html

[UDDI] *Universal Description, Discovery and Integration (UDDI)* [online] http://www.uddi.com/

[Vasu] Vasudevan, Venu; *A Web Service Primer*; [online] http://www.xml.com/lpt/a/2001/04/04/webservices/index.html

[W3Cws] *W3C Web Services Activity web site.* [online] http://www.w3.org/2002/ws/

[Waba] *Waba* [online] http://www.wabasoft.com

[White] White, James P.; Hemphill, David A.; *Java 2 Micro Edition* March 2002, Manning Publications Co.

[Wilson] Wilson, Brendon J.; *JXTA* ; Prentice Hall, 2002

[WSDL] *Web Service Definition Language (WSDL) Specification 1.0* [online] http://www.w3.org/TR/wsdl

[WV] *Wireless Village* [online] http://www.wireless-village.org/

[XML] *Extensible Markup Language (XML)* [online] http://www.w3.org/XML/

Appendix A

Additional packages for J2ME CLDC/MIDP

This appendix presents some of the packages under development for J2ME MIDP that is of interest in relation to network connectivity and possibilities. Today there are 37 Java Specification Requests related to J2ME, and most are intended as optional packages that will work with any profile, whether it is based on CLDC or CDC. An overview of all specifications in progress can be found at the Java Community Process site [JCP]. The process of how specifications are standardized was covered in last autumn's project [Lyng2001].

A.1 Location API for J2ME

The location specification [JSR-179] is proposed as an optional packages, targeted to work on CLDC 1.1 as a minimum and related profiles as well as profiles based on the CDC. Its intention is to enable developers to write mobile location-based applications, and the purpose it to provide a compact and generic API that produces information about the device's present physical location to Java applications. The API is intended to work with most positioning methods, such as GPS or E-OTD, but require that the device using the API supports some mechanism to determine its physical location. Security issues like access control and policy for permissions is necessary, but out of the scope of the API. MIDP 2.0's security model will provide one way to establish the required security framework.

The GSM 03.71 specification suggest a set of standards for implementing location services (LCS) on GSM mobile terminals and networks, and the 3GPP TS 071 specification and 3GPP TS 171 specification describe the same for the third-generation mobile devices.

A.2 Wireless Messaging

The purpose of the Wireless Messaging API [JSR-120] is to define a set of optional APIs that provides standard access to wireless communication resources so that 3rd party developers can build intelligent connected Java applications. The initiative for this JSR, which now is out for public review until the end of June, came from Siemens, and they also have the Specification Lead.

Wireless Messaging API (WMA) provides a basic MessageConnection and Message framework with general mechanisms for establishing a messaging application, which is independent of the underlying messaging protocol. Examples of such protocols are GSM Short Message Service and CDMA SMS. The architecture could also be used for receiving GSM Cell Broadcast short messages, which is a unidirectional data service where messages are broadcasted by a base station and received by every mobile station listening to that base station. [JSR-120]

A.3 Java Speech API

Java Speech API 2.0 [JSR-113] extends the work of Java Speech API 1.0, and is supposed to incorporate speech technology into user interfaces of Java programming language applets and applications. The Spec-

ification Lead is Conversational Computing Corporation, and the expert group was formed 2nd quarter of 2001.

The API is supposed to specify a cross-platform interface to support command and control recognizers and speech synthesizers, and consider future incorporation of dictation and other features. Especially application on embedded platforms will require speech, which will allow them to perform various speech related functions, for example speech recognition and text-to-speech. The specification itself will not provide any speech functionality, but access to speech functionality provided by supporting speech vendors through a set of APIs and event interfaces.

Target platform is J2ME, but whether it is both configurations, or just CDC is not specified. As the requirements to the target platform are access to sound resources and adequate computing resource, it might sound like it is intended for CDC and related profiles.

A.4 SIP API for J2ME

Session Initiation Protocol (SIP) [SIP] is an IETF standard signaling protocol that can be used to establish, modify and terminate sessions in IP network. The SIP specification abstract session from the physical network configuration and transfer to the application layer, and supports user mobility. An example of usage is to establish and manage multimedia IP sessions. SIP is by many predicted to become an important protocol in the future IP mobile phone environment, and SIP based applications will be essential enablers.

SIP is an IETF standard protocol for IP-communications, enabling IP-telephony gateways, client endpoints, PBXs and other communication systems or devices to communicate with each other. SIP primarily addresses the call setup and tear down mechanisms of sessions and is independent of the transmission of media streams between caller and callee. Several extensions to SIP are proposed like SIMPLE[1] [SIMPLE], an extension of SIP to support presence and instant messaging. The specification is the responsibility of the SIMPLE IETF working group.

The purpose of the *SIP API for J2ME* [JSR-180] specification is to enable SIP applications to be executed on memory limited, terminals, especially targeting mobile phones. The specification was proposed by Nokia Corporation and accepted in April 2002, and scheduled to be finished the second quarter of 2003. It will not make any assumptions on the standard and application used, and will be based on the SIP specification in the RFC2543 [RFC2543] and potential extensions required by IETF SIMPLE and SIPPING[2] [SIPPING] work. It will also take 3GPP requirements [3GPPsip] into account.

The specification is made possible by the introduction of TCP and UDP support in MIDP 2.0. Other requirement that must be included in MIDP is a more rigid security model, since the specification will base its work on the MIDP security model, and the Generic Connection Framework defined in CLDC. The target is to provide one API and prevent that there will become several different SIP APIs in the MIDP devices using TCP or UDP.

1. SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE)
2. Session Initiation Proposal Investigation (SIPPING)

Appendix B

Ad Hoc Network Technology

Peer-to-peer is often mixed with ad-hoc networks, and many claims it is the same phenomenon. One may say that peer-to-peer is a paradigm, and ad-hoc is a realisation of the paradigm where the peer-to-peer functionality is places in the lower layers of the protocol stack and therefore is dependent on the physical medium. Other peer-to-peer realisations might concentrate on making the implementation of the paradigm independent on the network technology, and so solves the problems at the application layer. The term peer-to-peer is mostly used about the latter

Ad hoc network is a network that forms without any central management, and consists of mobile nodes that uses a wireless interface to send packet data. The networks are self-organized and independent of infrastructure, and the organization and maintenance is influenced insignificant by the environment, but be handled by the users of the network. Self-organized networks are not intended as replacements for existing and future infrastructure-based networks, but as a complement in areas where cost, limitations and environment demands such a network. Both wired and wireless networks can be defined as "ad-hoc", but the wireless environment is considered the most interesting since the greatest challenges are found there.

A number of ad-hoc research projects are under development, like Terminodes, and an initiative has been taken to standardize ad-hoc protocols. This work is done by Mobile Ad-hoc Networking (MANET) Working Group, a working group inside the standardization body Internet Engineering Task Force (IETF). The goal is to develop specifications for routing in mobile ad-hoc networks, and introduce these as standards through IETF.

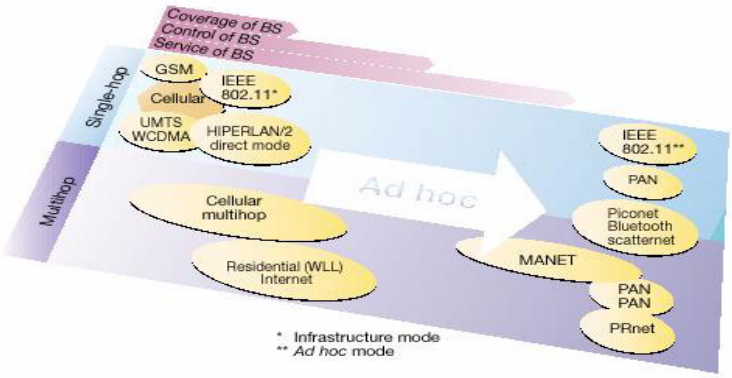


Figure B-1 Various wireless networks [Eric2000]

Generally speaking networks have evolved in a way where the responsibility of the infrastructure, security and application is left to the users of the network. This evolution will continue with the introduction of ad-hoc networks. To put things in perspective, ad hoc networks can be compared to traditional cellular and mobile networks. These are dependent upon an infrastructure since coverage is offered by base stations, and radio resources are managed from a central location, and services are integrated in the system.

As networks move away from central management, they evolve into ad hoc operations that could be single-hop or multi-hop. Multi-hop support in a network implies that packet data can be routed via other nodes in the network to reach the receiver, in contrast to single-hop networks where a node must have direct contact with the receiver of the data. WLAN is an example of a single-hop network, while PRnet is a multihop network. Figure B-1 shows some selected networks that are mapped to two independent aspects of an ad-hoc network. Horizontally direction shows the level of central control, while the vertical axis is divided after usage of radio multihopping.

Since ad-hoc networks consist of equivalent nodes that communicate over wireless links without central control, they will remove the limitations imposed by base stations and routers. Beyond this, the problem for ad-hoc networks are the same as for the traditionally cellular and wireless networks; optimalization of bandwidth, power control and improvement of transmission quality. The multihop nature of ad-hoc networks and the lack of fixed infrastructure produce new research work like announcement, discovery and maintenance. I addition there is addressing and self-routing.

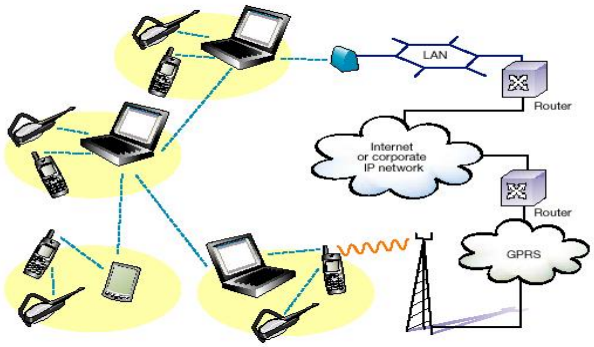


Figure B-2 Network of Personal Area Networks (PANs) [Eric2000]

Ad hoc networks with limited range can form Personal Area Networks (PANs) to improve communication between different mobile devices, and eliminate the need for cables. Mobile devices can be mobile phones and PDAs. The PANs can communicate between themselves, and have connection to Internet as shown in Figure B-2. Here, four Bluetooth PANs are connected via portable computers and Bluetooth links. Two of the PANs are connected to an IP-network, one via a LAN access point and the other via a GPRS/UMTS mobile phone. The advantages of PANs is that new technology can be incorporated quickly since they are able to support different access-technologies, all with ad-hoc functionality.

The intended use of ad-hoc networks is in areas where there is no available infrastructure, or the infrastructure is down as a result of an accident. Examples of such scenarios are rescue operations, or military operations in faraway places. In urban areas, ad-hoc networks can be used to faster utilization and extended coverage, and local sharing of documents. Figure B-3 shown an example of usage of ad hoc on an airport where PDAs, mobile devices and portable computers communicate with each other, and with access point to other networks like HiperLan/2 and WCDMA.

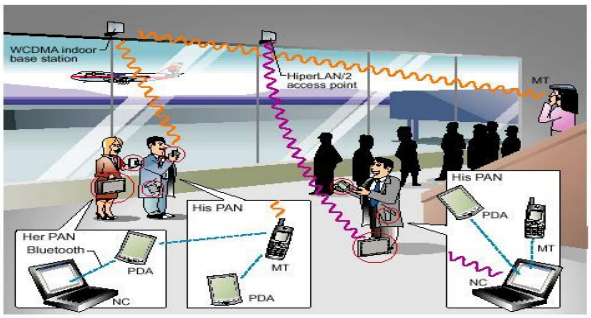


Figure B-3 Airport scenario [Eric2000]

Appendix C

ITSSystem UML diagrams and comments

This appendix contains some more information about the ITSSystem, like total view of the use case diagrams and the class diagrams, and all sequence- and collaboration-diagrams not already presented before. It also includes a discussion of the non-functional requirements that was not relevant for the evaulation of JXTA.

C.1 Use case

The use cases of the prototype are collected in three diagrams: one diagram for the public mode, one diagram for the use-cases related to private mode, and one diagram for the use cases that are considered common for both public and private mode. Login, exchange of messages and selection of modus are use-cases related to both modus, and these use-cases are presented in Figure C-1.

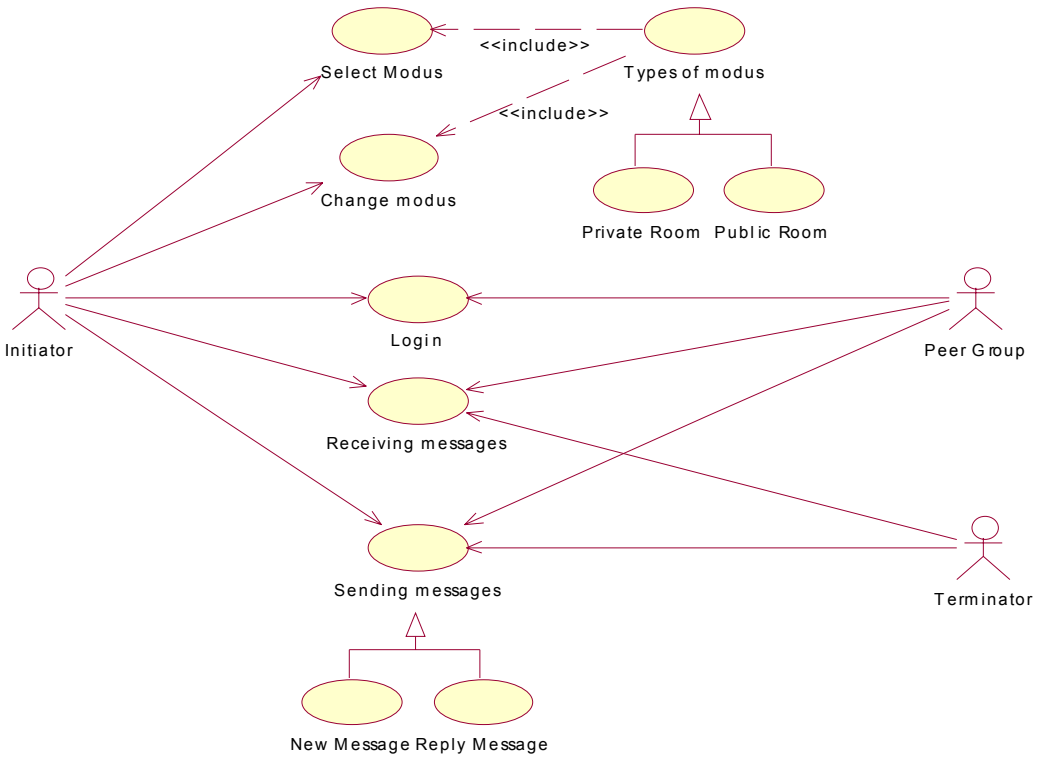


Figure C-1 Use case for login, exchange of messages and modus

C.1.1 Public room

The use cases specific to public room are related to the group list, joining of group, change of group and editing of the group list. Use-cases that are related to public room are shown in Figure C-2.

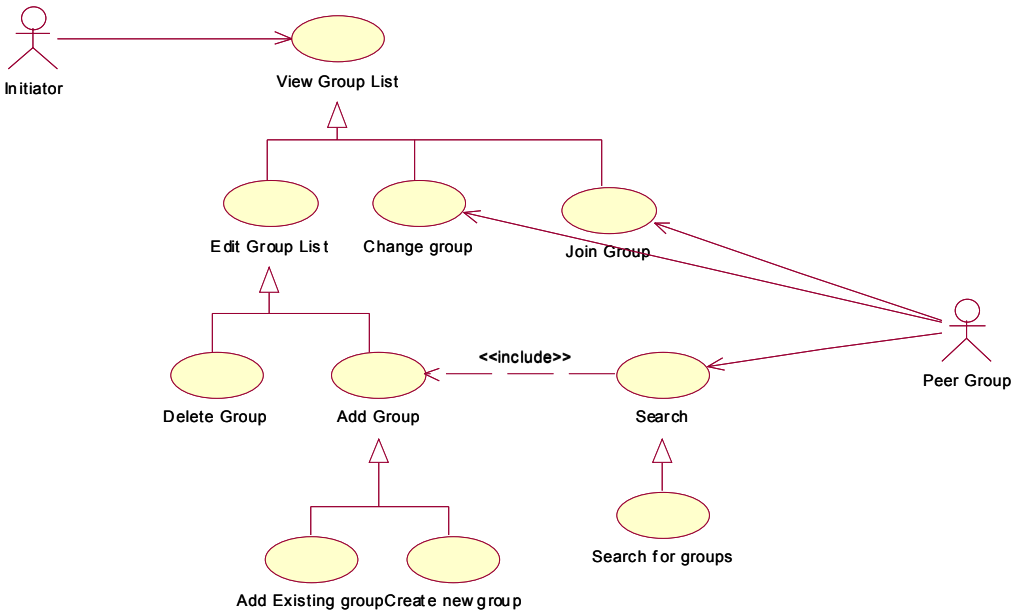


Figure C-2 Use case diagram for public room

C.1.2 Private room

The private room specific use-cases are related to the contact list, but also include the public room use-cases shown in Figure C-2. The additional use-cases for private mode are illustrated in Figure C-3.

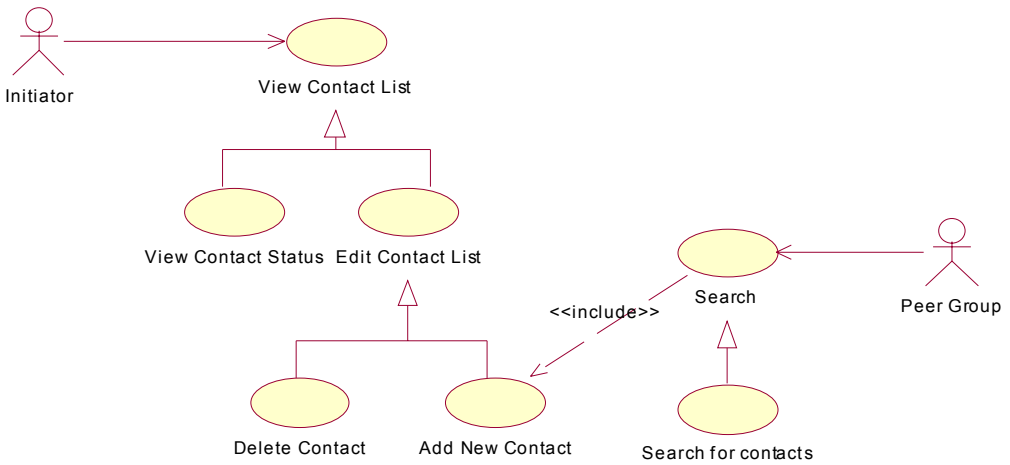


Figure C-3 Use case diagram for private room

C.2 Class diagram

The class diagrams of Chapter 7 display the public and protected variables and methods of the ITSSystem, which is the most interesting part when the purpose is to understand and get knowledge of the system. For people who want to further develop or maintain the system it is necessary to have an overview of the entire system with private variables and methods as well. This section shows the detailed class diagrams of the ITSSystem. An overview of the classes and interfaces in the system are illustrated in Figure C-4.

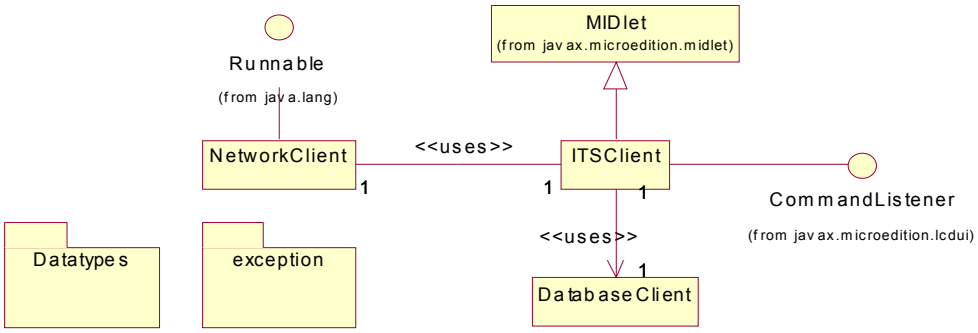


Figure C-4 Class diagram showing the overview of the ITSSystem classes

C.2.1 ITSClient

The class diagram of ITSClient with all the public, private and protected methods and some of the variables is shown in Figure C-5. Variables concerning the GUI elements like forms, buttons and text fields are left out.

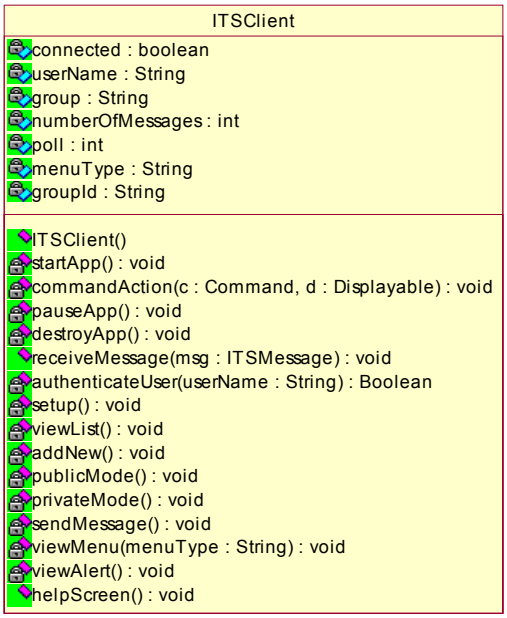


Figure C-5 Detailed class diagram for ITSClient

C.2.2 NetworkClient

Figure C-6 shows the detailed class diagram for the NetworkClient.

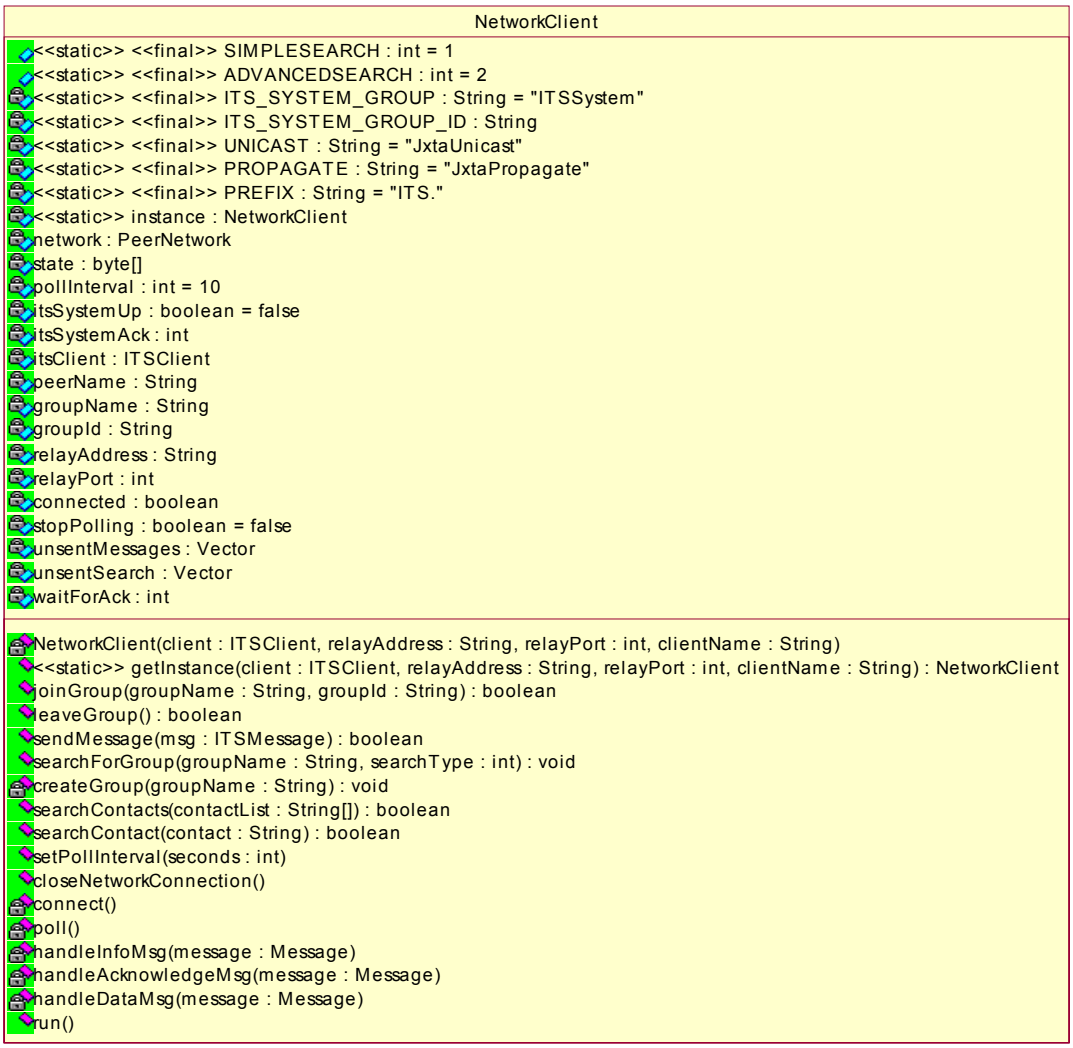


Figure C-6 Detailed class diagram for NetworkClient

C.2.3 DatabaseClient

Figure C-7 shows the detailed class diagram for the NetworkClient.

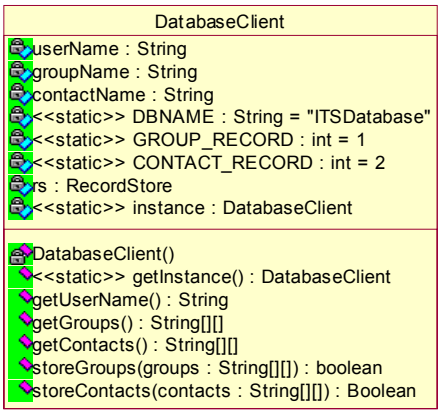


Figure C-7 Detailed class diagram for DatabaseClient

C.3 Sequence diagrams

Chapter 7 provided some of the sequence diagrams of the prototype. The rest are presented here, with some additional information and a more in detailed view. The diagrams will not be explained further since this is already done with the corresponding collaboration diagrams. The same goes for the collaboration diagrams presented in C.4 Figure C-8 is the corresponding sequence diagram to the collaboration diagram in Figure 7-21.

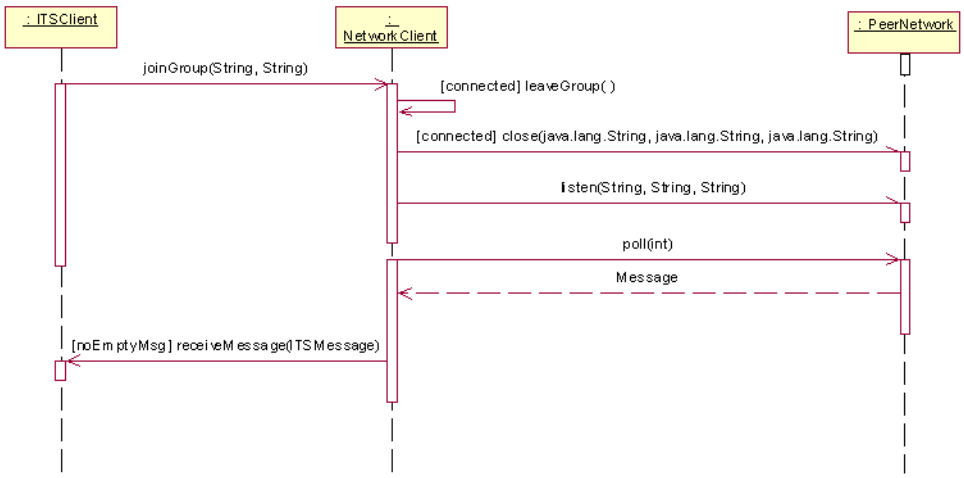


Figure C-8 Sequence diagram for joining an existing group

Figure C-9 represents the sequence diagram to the equivalent collaboration diagram in Figure 7-22.

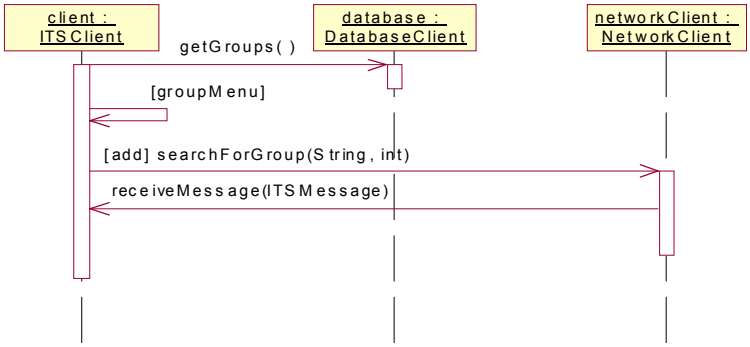


Figure C-9 Sequence diagram for viewing the group list and adding a new group

Figure C-10 is the corresponding sequence diagram to the collaboration diagram in Figure 7-24.

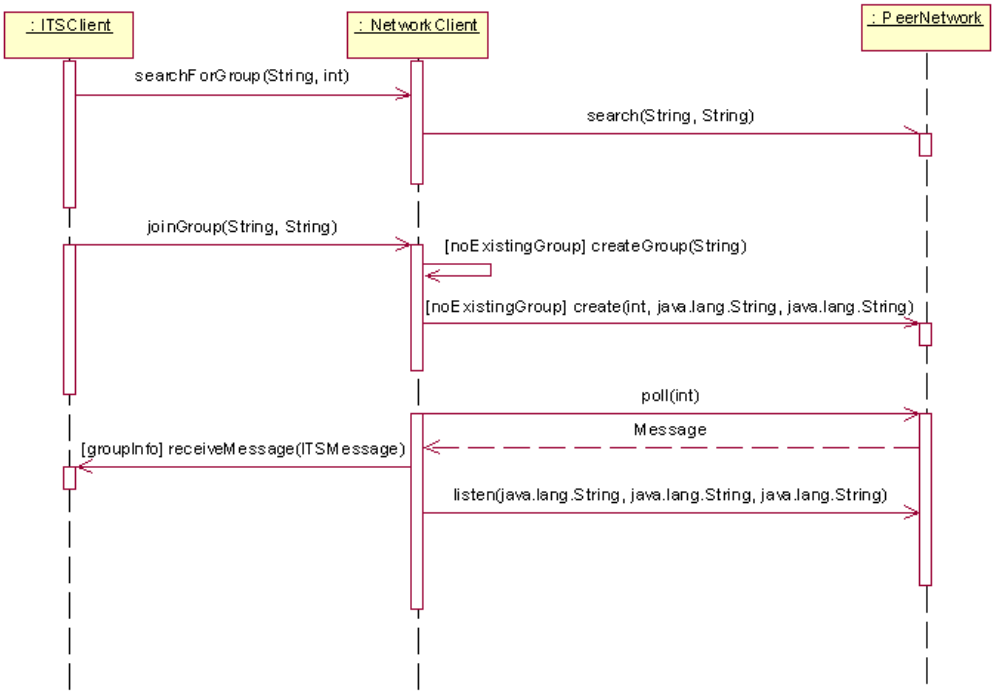


Figure C-10 Sequence diagram for adding a non-existing group

Figure C-11 and Figure C-12 constitute an extended version of the collaboration diagram for sending and receiving messages presented in Figure 7-26. What might be noted is that the data format used by the network is hidden from the ITSClient-instance. This is done by letting the ITSClient and the Network-Client exchange information using an internal data type called ITSMessage. NetworkClient fetches the information from the given ITSMessage and creates a Message to be sent to the PeerNetwork-instance.

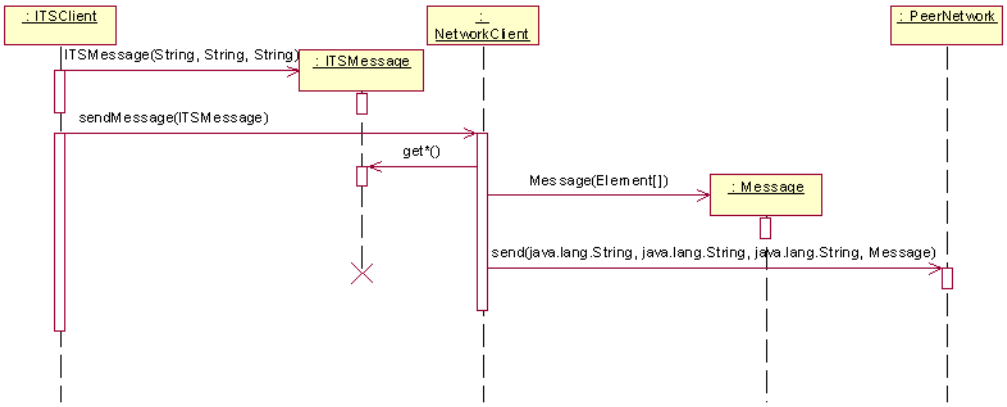


Figure C-11 Sequence diagram for sending a message

The same is the case when the NetworkClient receives a message from the network. The message format is changed and the ITSClient is given an ITSMessage containing necessary information and does not have to concern itself with the data exchange of the network.

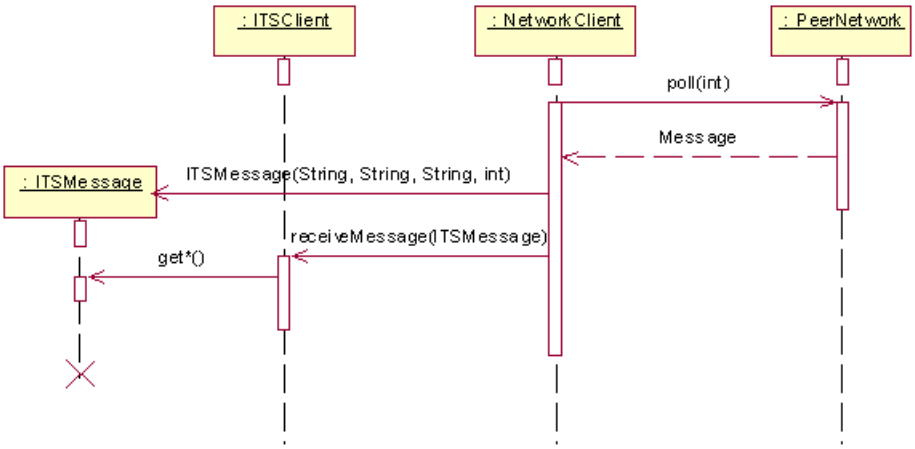


Figure C-12 Sequence diagram for receiving a message

C.4 Collaboration diagrams

This section presents the remaining collaboration diagrams of the ITSSystem. The diagrams will be shown with reference to the corresponding sequence diagram, as done in the previous section.

Figure C-13 shows the collaboration diagram of the corresponding sequence diagram shown in Figure 7-19.

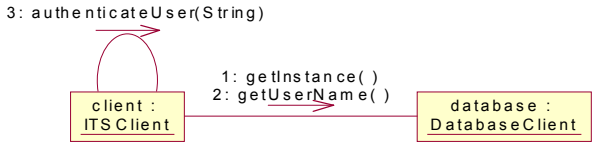


Figure C-13 Collaboration diagram for login and authentication

The diagram in Figure C-14 represents the collaboration diagram equivalent to the sequence diagram in Figure 7-20.

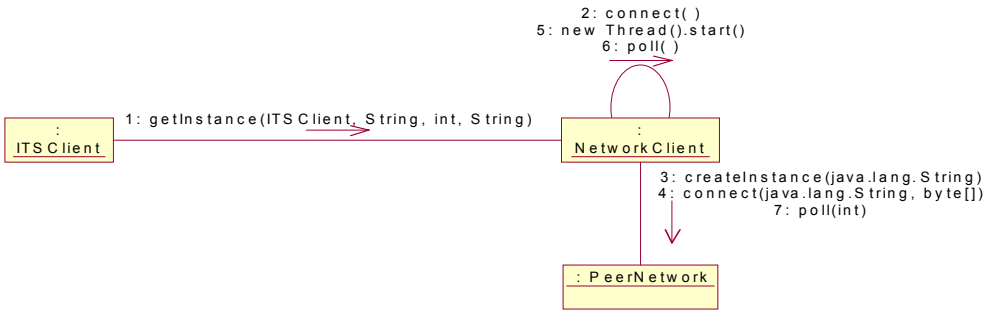


Figure C-14 Collaboration diagram for connecting to the network

Figure C-15 is equivalent to the sequence diagram shown in Figure 7-23.

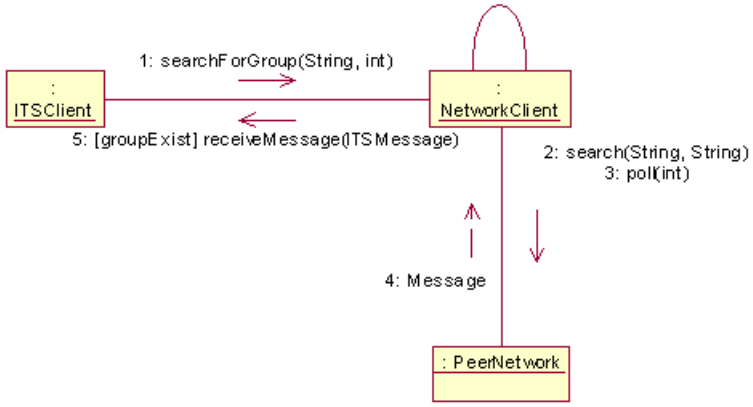


Figure C-15 Collaboration diagram for adding an existing group

Figure C-16 represents the collaboration diagram for the sequence diagram shown in Figure 7-25.

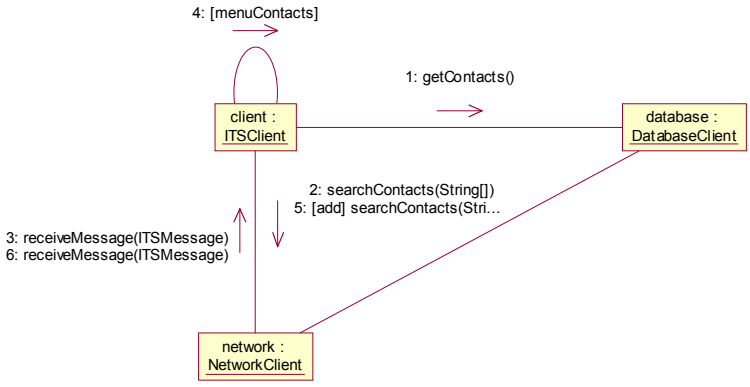


Figure C-16 Collaboration diagram for edit a contact list

The collaboration diagram in Figure C-17 represents the sequence diagram shown in Figure 7-27.

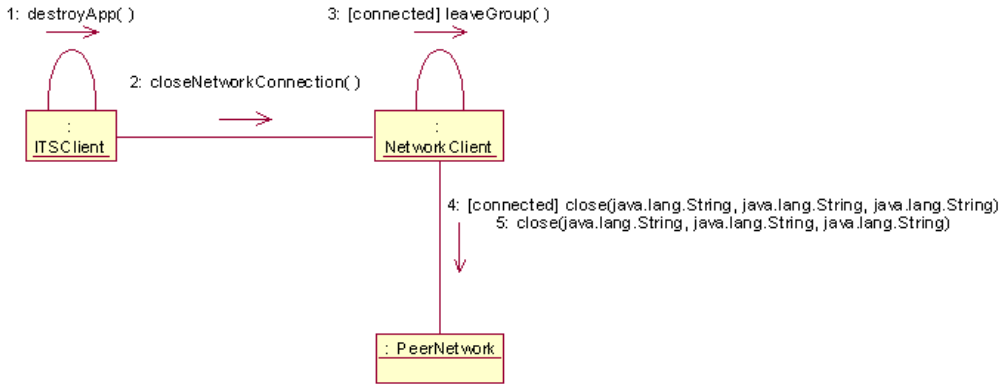


Figure C-17 Collaboration diagram for closing the connection and exit

C.5 Discussion of the non-functional requirements

This section provides a discussion of the non-functional requirments that are not relevant for the evaluation of JXTA.

Usability

NF - 4 states that a user should be able to use the service application without training, and should get to know it within 10 minutes. A developer designing an graphical user interface (GUI) for small devices must make it obvious for the user to understand, i.e. be userfriendly. In addition he must consider whether to make a solution that everyone can understand at once, without using a manual, or a solution that in the first place is not obvious to understand, but where the user can get the desired screen without too many steps. The user tests relieved that the GUI of ITSSystem is realtively easy to understand when you have used it for some time, but is in some cases not obvious to understand in the first place mostly because of the lack of confirmations. The best solution would surely be to make the program both very userfriendly and with a natural screen flow, which requires a limited number of steps for the user to execute what he desires.

Security

NF - 5 is concerned about the security of ITSSystem, especially according to private room. There are three security aspects that has to be taken into consideration: confidentiality, integrity and authentication. Confidentiality and integrity imply that information that is exchanged in the ITSSystem must be prevented from unauthorized listening or corruption of data. The way to do this is through encryption. The current version of JXTA for J2ME has no security built in and J2ME MIDP has no security mechanisms for encryption. As a result the current version of the ITSSystem cannot prevent unauthorized parties from snooping. But packages like Bouncy Castle, an open source cryptography package, could be included to ensure privacy until further security is provided in J2ME. Authentication could be done using a username and password, but the problem is who is going to verify that a party is who he claims to be.

Cost

NF - 8 covers the possibility for the user to control the cost usage, i.e. the money spent. ITSSystem 1.0 realize this by setting the poll frequency. In order for this to influence the costs, the billing of the network usage must be according to the amount of data sent, not the time the user is online. This is decided by

the access network used to connect the mobile device to a network. As of today, the leading cellular netowork is GSM, which is circuit switched and charge according to time. GPRS and 3G networks on the other hand are packet switched, and charges according to amount of data sent. The prices for GPRS data are too high, and should be notably reduced if a data service like ITSSystem will be worth using. Letting the user set the polling frequency can result drawbacks like unneccessary use of bandwitdth. An additional functionality could be that the application detected how often messages arrived to the specific user, and that the polling frequency was set based on this information and on the current network load.

Appendix D

ITSSystem User Manual

The Intelligent Transport Service System (ITSSystem) 1.0 is a communication service for travelling people. You can get in touch with other users "on the road" being in the same area or travelling the same distance as you are. The goal of the service is to get the most out of your trip, and not let queues, or roadwork, etc. get in the way. You can achieve this by participating in a group, which represents an area or a distance, and exchange information with the other members of this group. Each group represents a chatting room, and you will receive every message sent to this group. You can keep a list of your groups locally, and edit this list whenever you want. This document will help you to understand the functionality of the service, and describes how to use it.

D.1 Getting started

The ITSSystem 1.0 is only about 25Kbyte and can be downloaded to any J2ME compatible mobile device. How the program is loaded onto the device and how the program is found, started, stopped, and removed from the device depends on the target device and the operating system (OS) used.

The ITSSystem consists of two files: ITSSystem.jad and ITSSystem.jar. The JAD (Java application descriptor) file describes, among other things, the size of the JAR-file, and should be loaded into the device first, to check whether a full load of the JAR file is worthwhile. The JAR-file contains the program and should be loaded onto the terminal in order to run the ITSSystem.

When you have installed the program and got a username, you can start the ITSSystem. You will first get the screen to the right. Log in using your username and click OK. EXIT closes the program. Editing the setup is explained in part D.4.

D.2 The group list

The first screen that appears after login (if you choose not to edit setup) is the group list pictured to the left. The group list is a list of the groups you might want to participate in. The first time you log in this list is empty and you must add the groups you are interested in. When you have added groups to your list you can select one of them to join. This is done from the group list shown to the left. Click OK to join the group you have selected. If you click the MENU button, the group-editing menu will appear.

D.2.1 Edit group list

The menu shown to the right is the group-editing menu. From this menu you can add new groups or delete existing groups. If you choose Join group, the group list will appear. In addition you can enter the setup from this menu. CANCEL will bring you back to the group list.

D.2.2 Add group

To add a new group, choose "Add group" from the group editing menu and click OK. The screen shown at the left will then appear. To add groups to the list you must either know the group name of an existing group or add a new group. Enter the group name in the text field, and click OK, to add a new group to the group list. The updated group list will then appear. CANCEL will bring you back to the menu.

D.2.3 Delete group

To delete groups from the list, you choose "Delete group" from the menu. Then the group list will appear, and you must select the group you want to delete, and click OK. The same screen will reappear after deleting the group, with an updated group list where the deleted group has been removed. CANCEL will bring you back to the menu without deleting any of the groups.

D.3 Public communication

ITSSystem 1.0 includes the possibility to communicate with the whole group you are participating in. (Private communication with only one other user is not supported in this version.)

After joining a group you will receive all the messages sent to this group, and you can send messages to the group. The messages you send and receive will be printed to the screen together with the username. The number of messages viewed at a time is at default set to three, but can be changed in the setup. The figure to the left shows the public communication screen.

D.3.1 Create and send a message

To create a new message to send, click the SEND button. The screen to the right will then appear. Write your message in the text field and click SEND to distribute the message to the group. CANCEL will bring you back to the public communication screen without sending the message.

D.3.2 Change group

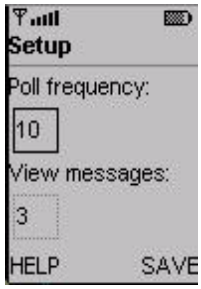
If you want to connect to another group you have to select the "Change group" option from the menu. The menu to the left appears if you click MENU from the public communication screen. “Change group” will disconnect you from the current group and bring you to the group list were you can choose another group. You can also choose to add a new group to join. This is achieved by choosing “Add group” from the menu and you will then enter the “Add new” screen. After adding a group to the list, the group list will appear and you can choose to join the newly created group or one of the other groups.

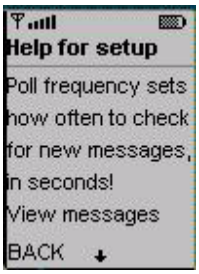
You can also edit the setup from this screen, and return to the menu after saving the setup. If you click CANCEL you will return to the public communication screen, and you will still be participating in the same group. The same thing will happen if you choose the “Back to <group name>” option.

D.4 Setup

You can edit the default setup values by selecting “Setup” in one of the menus or by choosing to edit the setup in the login screen.

In this version of ITSSystem there are two values that can be set in the setup: The polling frequency, and the number of messages viewed in the public communication screen. The polling frequency decides how often to ask for messages from the network. As default the poll frequency is set to 10 seconds. If you are planning to have an ongoing chat, you may want to set this field to a lower value. On the other hand, if you don’t need to get messages very often, and are not planning to send too many messages either, you can spare the network for much traffic (and your self for much money) if you set it to a higher value. The number of messages is set to three as default.



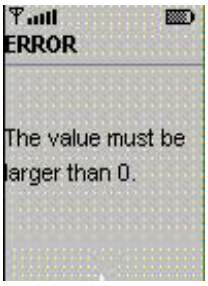


SAVE will change the setup and you will return to the previous screen if you entered setup from one of the menus, otherwise you will get the group list screen.

If you click HELP the screen to the left will appear. This screen will give you information about the fields in the setup and you can return to the setup by clicking. BACK.

D.5 Alert messages

If something goes wrong or you fill in invalid values, an error message like the one to the left will appear. This will be viewed for some seconds before it automatically closes and returns to the previous screen. If you could not connect to the network or could not get access to the database containing your group list, the program will close since the program depends on the network and the database to function.



Appendix E

3rd party software development tools

Different tools exist to develop J2ME CLDC/MIDP applications and get it running on an iPaq. Some of these tools are presented shortly in this appendix.

Sun has provided a toolkit for help developers in the implementation, and some companies have developed virtual machines for the iPaq. Among other Insignias Jeode platform and IBM's j9 virtual machine included in their WebSphere development environment.

E.1 Java 2 Micro Edition Wireless Toolkit

Java 2 Micro Edition Wireless Toolkit [J2MEwtk] offers a lightweight way of building and testing a MIDP project. It comes with the CLDC and MIDP libraries, utilities, javadoc documentation and example MIDlets. The utilities include emulators for different mobile phones and an easy way to edit the JAD-file and the properties included in the file. Some of the emulators are shown in Figure E-1, with the wireless toolkit in the background.

The current version of the wireless toolkit is based on the technical specifications of J2ME Connected Limited Device Configuration (CLDC) 1.0 and of Mobile Information Device Profile (MIDP) 1.0. The wireless toolkit runs on any Java 1.3 enabled platform, and can be downloaded and used free of charge from java.sun.com.

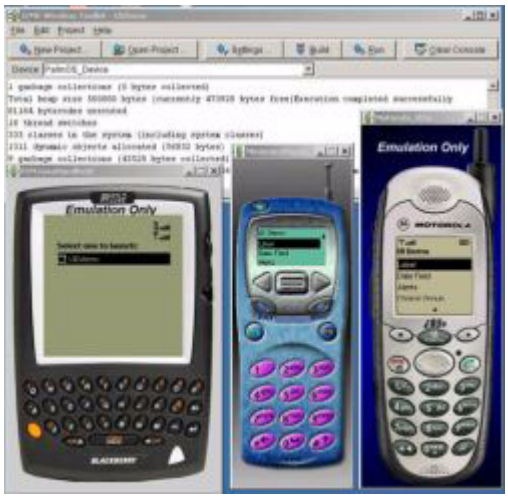


Figure E-1 Screenshot of the wireless toolkit from Sun [J2MEwtk]

Optionally, the toolkit can plug into Sun's Forte for Java Community Edition IDE [Forte]. Forte is a development environment from Sun, and the Java Community Edition can be downloaded and used free

of charge. Doing so, makes the developer able to develop and test applications from start to finish, but the developer is free to choose development tools since the toolkit is only capable of compiling, running, and packaging the application.

E.2 Compaq iPaq Pocket PC

An iPaq Pocket PC is a powerful, mobile device from Compaq[iPaq]. They are powered with Microsoft Windows, and are not equipped with any development tools or platforms for running Java applications.

Figure E-2 Compaq iPaq

An iPaq can use different PCMCIA cards by usage of an expansion pack that holds the cards, that can be attach to the iPaq. This is useful to get the iPaq online by using a WLAN card, or a GPRS card, which is of interest in connection with MIDP application development. Other cards used could be a GPS card to determine the current location of the iPaq.

IPaq are powerful handheld devices, and are able to run full Java 1.3 under Linux. Most attempts to place Java on the iPaq have resulted in support for Personal Java [PJava] and lately also CDC and the Personal Profile, the new version of PersonalJava included in J2ME, under development. But there are some possible solutions to run a kvm for MIDP on an iPaq:

- Run the PersonalJava virtual machine from Sun, supporting version 1.1, and supply with the me4se emulator.
- Run the virtual machine from Insignia, the Jeode platform, which support version 1.2 of PersonalJava, i.e. version 1.1.8 of the standard edition of java. On top of this virtual machine, supply the me4se emulator.
- Using the kvm for Pocket PC running an ARM processor, included in WebSphere Micro Environment from IBM.

The two next sections will introduce the Jeode solution and the IBM-solution.

E.3 The Jeode platform

The Jeode platform is developed by Insignia Solutions, a company that seeks to become “a key provider of provisioning-infrastructure software that focuses on the technologies needed for wireless carriers, handset manufacturers, and others in the Java eco-system to fully capitalize on the Java standard” [Insignia].

The Jeode platform is an independent implementation of a Java virtual machine, both for PersonalJava 1.2 and EmbeddedJava 1.0.3, and is roughly equivalent to JDK version 1.1.8. The PDA edition is tailored for Pocket PCs, PDAs and related handheld devices, is compatible with PersonalJava 1.2, and support both Linux and Windows CE. It can be bought online for about $20.

The disadvantage with the Jeode solution is that the virtual machine is rather big, around 5 Mb, where the emulator described next uses a small fragment.

E.3.1 me4se - Micro Edition for Standard Edition

ME4SE [me4se], micro edition for standard edition, is a project which purpose is to make J2ME APIs like the user interface (LCDUI) and the network packages (the Generic Connection Framework) available for the standard edition of java (J2SE).

The motivation is to enable some limited development support for platforms where no emulator is available, and allow demonstration of MIDlets before installation on the device. It should also enable PersonalJava devices to run MIDlets, which is of interest for the iPaq at disposal when developing the prototype.

Distribution of me4se is under the GNU Public License [GNUpl], and the package is developed by a theme of independent software developers. Me4se uses the png library from sixlegs.com for PNG support [Pnglib], which must be downloaded and installed with the me4se package.

Using me4se an application using JXTA for J2ME and MIPD will execute on the iPaq, as long the application does not try to establish a network connection.

Drawbacks for the Jeode plus me4se solution is that it is a rather troublesome process to have get a MIDlet to execute, and that the me4se is still under development and might not support all the features in MIDP yet.

E.4 WebSphere Micro Environment

IBM has renamed its VisualAge to WebSphere [IBM], and included a development environment for J2ME applications, supporting both CLDC/MIDP and CDC with additional profiles, called WebSphere Micro Environment. The environment includes an emulator, debugging and creation of executables required by their own virtual machine, j9.

WebSphere Micro Environment is companied by a runtime environment for a wide area of devices [IBMrt], among other the PocketPC run on an ARM processor, i.e. an iPaq. Their virtual machine is an extended version of the J9, the virtual machine first developed with the Palm as target, to support J2ME.

An evaluation version of Device Developer with Micro Environment is available for free from IBM at the moment, but it is time consuming to get to know the program and find out the necessary steps to take to create a MIDlet since the documentation is insufficient. The same counts for the runtime environment on the pocket pc. Except from this, it seems like a good solution for a virtual machine. The platform takes minimal of space since a user can decides and pick the programs and libraries he wishes, and so reduce the virtual machine to a minimum only capable of running applications. If the user later decides that he wants to develop application on the iPaq, he only installs the necessary libraries. The drawback is that it is not easy to understand what to install and how to run the application.

Appendix F

Presence and Instant Messaging

Presence and Instant Messaging is an old concept in the computer software industry. Many software and proprietary protocols have been developed to support exchange of messages between users, both in the computer network environment, the telephony world. But most of these solutions are not interoperable. Lately several initiatives have been made to create a standard protocol for presence and instant messaging.

Presence is the notion of an entity being a part of a network. The entity can be for example a mobile device, a PC, or a telephone. Concepts important to Presence would be when an entity enters and exits a network, and relevant information about the entity such as location, duration, relationship to other entities, and similar. Instant messaging is the action of exchanging messages between two entities in a network. The capabilities included are sending and receiving messages, opening and closing messaging sessions, resuming messaging sessions, and other actions relevant for message exchange.

Presence and instant messaging protocols that have been devised or that are currently being designed is Wireless Village [WV], SIMPLE[1], Jabber, AIM, MSN and Yahoo. Some are intended for legacy networks, and others for IP telephony networks. Jabber is an open, XML-based protocol. Multiple implementations exist, and these have mainly been used to provide instant messaging and presence service. American Online's AOL Instant Messenger (AIM) is beginning to be provided as a service for mobile users. The Wireless Village standardization effort is targeted at the mobile marked, while SIMPLE is targeting IP telephony networks. Both Wireless Village and SIMPLE is covered in more detail later in this appendix.

JAIN [JAIN] is a set of API's that aims at enabling the integration of the Internet and Intelligent Network (IN) protocols. This is referred to as Integrated Networks. The initiative consists of two areas of development:

- The **Protocol API Specification** specify interfaces to wire line, wireless, and IP signalling
- The **Application API Specification** address the APIs required for service creation within a Java framework spanning across all protocols covered by the Protocol API Specification.

All specifications aimed at integrated networks are part of JAIN. This includes the presence and instant messaging packages that are being standardized at the moment through the Java Community Process. Four of these specifications are covered in later sections of this appendix, two in relation to SIMPLE and two in relation to generic presence and instant messaging protocols.

1. SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE)

F.1 Wireless Village

Wireless Village is an industrial consortium founded by Ericsson, Motorola, and Nokia. The initiative started in the beginning of 2001, and was publicly launched in April 2001. Initial specification is expected to be published by the end of 2001.

The objective is to establish a common mobile Instant Messaging and Presence Service (IMPS), specifically optimized for the mobile environment. The solution is intended to be transport independent, i.e. capable of being used over many different data protocols in use by wireless and networked applications. For this reason, Wireless Village bases the specification on XML and aims at making it easy to expand. Examples of protocols that can be used are Short Message Service (SMS), Multimedia Messaging Service (MMS), GPRS and SIP, but initially the specification will support SMS and MMS. The specification does neither specify the user interface or restricts the types of device that the product can support, thus making is easier for vendors to incorporate in existing solutions.

Communication between users range for simple text messages, to rich, multimedia content-based messages. The representation and structure of data on the device or within the networked data repository will not be specified by the protocols, but they will describe how IMPS protocol data is represented over the network.

F.2 SIMPLE

Session Initiation Protocol (SIP) [SIP] is an IETF standard signalling protocol that can be used to establish, modify and terminate sessions in IP network. It enables IP-telephony gateways, client endpoints, PBXs and other communication systems or devices to communicate with each other. An example of usage is to establish and manage multimedia IP sessions. SIP is by many predicted to become an important protocol in the future IP mobile phone environment, and SIP based applications will be essential enablers.

SIMPLE [SIMPLE] is a set of natural extensions made to the SIP protocol to support Presence and Instant Messaging. The specification is the responsibility of the IETF working group SIMPLE.

F.2.1 JAIN SIMPLE Presence and JAIN SIMPLE Instant Messaging

Two specifications are undergoing the Java Community Process to become a standard API supporting presence and instant messaging in SIMPLE. These are the JAIN SIMPLE Presence [JSR-164] and JAIN SIMPLE Instant Messaging [JSR-165], that differs from the SIP API for J2ME covered in appendix A in that the latter is platform agnostic and the former are not.

The specifications are covered under JAIN, which is Java's packages for communication across Internet and Intelligent Networks, and target at both the J2ME platform and the J2SE platform. JAIN SIP API supports the SIP protocol, and is dedicated to Java session control in the telecommunication and Internet industry. JAIN SIP Lite is an abstract JAIN API to the SIP protocol that is dedicated to user agents running on both the J2SE and J2ME protocol. JAIN SIP Lite is a lightweight definition of SIP for user agents, including J2ME devices.

JAIN SIMPLE Presence and JAIN SIMPLE Instant Messaging are both proposed by Panasonic Information and Network Technologies Laboratory, and were accepted as specifications in the Java Community Process in January 2002. The goal is to create an API that is independent on a specific SIP/SIMPLE vendor, and to allow integration with other JAIN solutions.

JAIN SIMPLE Presence provides a standard portable and secure interface to manipulate presence information between a SIMPLE client (watcher) and a presence server (presence agent). Capabilities that are needed to support presence are presence servers (receiving subscription requests, authentication and au-

thorizing requests, sending notifications, read/write presence information, etc.) and presence clients (buddy and buddy list manipulations, sending subscriptions, receiving notifications, etc.).

JAIN SIMPLE Instant Messaging provides a standard portable and secure interface to exchange messages between SIMPLE clients. Instant messaging capabilities are sending messages, receiving messages, opening and closing messaging sessions, resuming messaging sessions, etc.

F.3 Generic presence and instant messaging APIs for J2ME

There exist a multitude of messaging protocols, as well transport protocols. For the user it would be of great value to have one API to deal with, and not be concerned of the underlying protocols.

Two APIs are under development to support a protocol agnostic API for instant messaging and presence, JAIN API for Instant Messaging [JSR-187] and JAIN API for Presence [JSR-186]. The former is intended to provide a standard framework for developing and deploying new Java instant messaging applications and services, while the latter focuses on a framework for presence. Both frameworks will require no prior knowledge of the underlying protocol, which could be for example SMS, MMS, WAP, WSP, HTTP or HTTPS. In addition, the applications should be interoperable, i.e. able to run over a wide variety of protocols such as Wireless Village, SIMPLE, Jabber, AIM, MSN, and Yahoo.

The presence specification covers presence from the entity, that is, when and how a device enters an exits a network, and addresses the interfaces a client require to communicate with a Presence server. Java specification JSR 123 Presence, Availability, and Management (JAIN PAM) addresses the needs and concerns of presence for a server within a network.

Both specifications are proposed by Panasonic Information and Network Technologies Laboratory, and accepted in the Java Community Process in May 2002. The work is estimated to take up to three years from the time the Expert Group start its work.