Framework for collaboration in mobile ad hoc networks

Brouwer, R.J.

Award date:
2001

TECHNISCHE UNIVERSITEIT EINDHOVEN

Department of Mathematics and Computing Science

MASTER'S THESIS

Framework for Collaboration in Mobile Ad Hoc Networks

by

R.J. Brouwer

August 2001

Supervisor: prof. dr. P.M.E. De Bra (Technische Universiteit Eindhoven)

Advisor(s): dr. M.R.V. Chaudron (Technische Universiteit Eindhoven)
H. Fagrell Ph.D. (Newmad Technologies AB)

ABSTRACT

Recent advances in hardware have resulted in the development of small wearable terminals (PDAs) with networking capabilities. While currently these terminals are merely used as schedulers and calculators, the technologies of today could offer much more sophisticated use. As users move away from traditional desktop computing environments and move towards mobile and ubiquitous computing environments, there is a greater need for applications that support these environments in which users and devices move around and make use of computational power that is only temporarily available.

The objective in this thesis is to make it easier for developers to design services for such settings. Accordingly, the research question is:

How can we develop a framework that assists application developers in designing services for PDAs connected in Mobile Ad Hoc Networks?

The question is pursued by designing a flexible framework, capable of running on different types of devices. It addresses many of the general problems that developers would run into. In addition, an implementation of the framework called "TransPeer" is discussed, along with a number of applications that have been built with it.

This thesis adds to the body of research by identifying and solving relevant issues in this new and compelling domain.

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank some people that have supported me in completing this thesis:

First of all, I would like to thank my project leader Henrik Fagrell from Newmad Technologies AB. You have put a pressure on me that fit me. Carefully selecting the words to coach me, you managed to keep me focused on my thesis and I thank you for doing so.

Secondly, I would like to thank Jonas Larsson (Newmad Technologies AB) for showing me the art of programming. Due to your help and knowledge my memory has no leaks any longer.

Thirdly, I would like to thank both Fredrik Ljungberg and Per Dahlberg (both from Newmad Technologies AB) for giving me this opportunity in the first place. It must have been a tough decision to allow a Dutchman in a Swedish company. It was a pleasure working with you both during my thesis.

Last but not least, I would like to thank my teacher Prof. Dr. Paul De Bra. The communication between us was minimal, yet gave me an enormous freedom to do research. Thank you for giving me this freedom to do it "my way".

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION AND MOTIVATION

In 1993, Apple Computer Inc. introduced the world to the first PDA, the Newton. They were dubbed PDAs (personal digital assistants) by John Sculley, former chairman of Apple Computer Inc, and were sold as the ultimate information appliance. Sculley predicted PDAs would become ubiquitous tools that would hold telephone numbers, keep your calendar, store notes, plus send and receive data wirelessly. However, the Newton was not able to deliver all of those features at the time it was released.

For the next three years, PDA sales dwindled, and were almost off the charts.

Then, in March 1996, Palm Inc. delivered the industry's first truly compelling handheld computer, the PalmPilot. A robust yet small go-anywhere device that helped people manage and organize their personal and professional lives by providing instant, anytime access to schedules, important phone numbers, to-do lists and other key information. This new type of information management was met with tremendous acceptance. Mobile, busy people embraced the small and powerful Palm handhelds.

Today, Sculley's predictions have come true in the form of a wide variety of handhelds all capable of the promised and more. The latest advances in wireless technologies have opened the doors to the next step in the evolution of the PDA. The next-generation telecom networks (3G) will enable global connectivity, while the latest datacom networks such as Bluetooth and Wireless-LAN (IEEE 802.11b) will enable local, short-range connectivity. Adoption of these technologies in PDAs is under way, and is expected to reach maturity between the years 2003 and 2005.

However, while these wireless network technologies are rapidly evolving the applications that get most attention today are the lightweight version of PC applications. Without the capability to transfer and synchronize data back to a desktop system, there's little benefit in having a word processor or similar feature on a PDA. Its no surprise then that this is a feature that has improved significantly in recent years, not in the least thanks to the efforts of third-parties who have developed both hardware accessories (e.g. keyboard) for use with PDA docking cradles and software applications designed to make the synchronization task as comprehensive and as simple to execute as possible. As the synchronization between PCs and PDAs has matured, the next generation of applications will focus on using the wireless network technologies soon to be available.

While the computational power of PDAs is roughly three years behind that of desktop PCs, the settings in which they are used differ from the traditional desktop. Unlike a desktop PC, PDAs are used in settings that are highly dynamic. It is in these unpredictable settings where wireless network technologies can take the PDA to a higher level by providing connectivity to the outside world. Not only simple applications such as a shared white board or chat, also more advanced applications like a trading agent for auctions can facilitate the user in their everyday tasks. The ability to share information between PDAs and sharing this information with the user, or make decisions based upon this information opens the door for new types of applications for the PDA.

These settings where applications need to communicate and collaborate could benefit from a framework enabling this collaboration. Easy access to other members of the network is provided and will result in a lower "time to market" for many applications. This thesis presents a framework for collaboration in mobile ad hoc networks. In addition to the implementation of the framework, some applications using the framework are presented.

1.1 A Mobile World

Mobile Ad Hoc Networking is a name given to a technology under development for the past 20 years. A Mobile Ad Hoc NETwork (MANET) consists of mobile platforms, which are free to move arbitrarily [5]. MANETs pose some new challenges not seen in traditional fixed infrastructure networks. A major issue in these networks is routing, since Mobile Ad Hoc networks cannot rely on specialized routers for path discovery and traffic routing. MANETs are discussed in detail in chapter 2.

Two example applications presented in this section will show the strength of collaboration in Mobile Ad Hoc Networks.

The museum guide
When people enter an art museum they often buy a booklet that contains information about the paintings, the location of the different paintings and some background information about the painters presented in the museum. The bigger and well-known museums sometimes offer their visitors a portable CD player that gives the information through speech. The information presented is a subset of the information available and is enough for the average visitor. However, in some cases people want to learn more about some painting or painter. Another issue is the burden of pausing the CD player and finding the right place to start the player when skipping a painting

The solution to these issues could be by using a PDA with networking capabilities. In the future, the visitors borrow a PDA (or use their own) when entering the museum. The exploration of the different paintings starts from here. Whether they want a quick tour around the paintings of Rembrandt or a full tour around the museum, the PDA can be their guide. Based upon some profile, the museum can offer their visitors a more personalized tour around the different paintings. When customers get close to a painting they will get some information on their PDA about the painter, the background information of the painting or even read the messages of other visitors of the museum. If they wish more information about some topic it can be requested via their PDA. The visitors can access an almost unlimited resource of information through their PDA.

What is needed for this museum guide is a wireless network and clients that can access the local network. To provide the visitors with local information (e.g. information about a painting they are watching) multiple "hotspots" could be set up in the museum. Whenever a client comes in reach of the hotspot, localized information can be provided. The software will have to enable this collaboration between the hotspot and the client.

The conference suite
A typical example where people are constantly interacting is at a meeting or conference. Different thoughts are discussed, possibly presented on a white board or beamer. Notes are taken and appointments are made for the next meeting. Currently the majority of the attendees uses either a (paper) notepad, a laptop and in some cases a PDA to write down their notes. The following applications offer added value to meetings, enabled by PDAs:

- A shared whiteboard. Users can draw text and graphics on their PDA while others can see and react to these drawings immediately. Even though a real whiteboard should be used in some scenarios, the PDA can be used to quickly draw some thoughts and share them.
- A shared agenda. The software will automatically look for the dates when all the attendees are available.
- A presentation viewer. It depends on the speed of the host how well people understand some slides. If attendees wish to look at a slide a little longer in order to get a better view on the topic, they can easily browse the presentation using their PDA.

The proposals mentioned above are only some of the examples where a PDA could help increase the quality of the conference or meeting.

These applications need a Mobile Ad Hoc Network. Often the attendees can't rely on an existing infrastructure. Even if there is an existing infrastructure it might not be possible to join it. Furthermore, software will be needed to realize the proposed applications.

1.2 Framework for Collaboration

After showing the potentials of Mobile Ad Hoc Networks, this section will discuss the need for a framework for collaboration and explain what is meant by a collaboration framework in the context of this thesis.

1.2.1 Definitions

The word "framework" can mean several things. In this thesis, it refers to the foundation on which applications can be build and run.

The word "collaboration" means in essence a joint effort in order to accomplish a goal. The framework will enable this collaboration between applications.

A definition of a framework for collaboration in Mobile Ad Hoc Networks can now be presented as:

> *A Service that enables entities (applications) that are part of a Mobile Ad Hoc Network, to work together to reach their desired goal*

1.2.2 Benefits of a Framework

Today many of the multi-user applications design and implement their own communication layer of an application. Many application developers start from scratch when building these applications. A set of APIs might be used, but these often to not provide more than a wrapper around the bare libraries available. Especially when it comes to collaborative applications, the networking part of the development phase can take some time. Therefore, a framework providing easy access to the network could lower the time-to-market drastically. Developers can focus on their own piece of software, not having to worry about the network, reuse in one word. Another advantage of having a framework is stability. A well-designed framework takes time, but once it is stable it can be used over and over again, with known results. A last advantage is the easier maintenance of both the framework and applications using the framework. Due to separation of concerns, redesign or updates of either the framework or the applications using the framework can be done without affecting the other.

1.3 Thesis Research Question and Contribution

Section 1.2 gave some of the benefits of a framework. Reuse, maintenance and well-known results were mentioned. Therefore, the goal of this thesis is to support reuse and make it easier to build and evolve collaborative applications.

The research question of this thesis is:

> *How can we develop a framework that assists application developers in designing services for PDAs connected in Mobile Ad Hoc Networks?*

The expected contributions of this thesis are:

- Identification of requirements to support the building and evolution of collaborative applications.
- A framework that both "lowers the floor" (i.e. makes it easier for designers to build applications) and "raises the ceiling" (i.e. increases the ability of designers to build more sophisticated applications) in terms of providing this support.
- Implementation of this framework.
- Building a number of applications that use the framework.

1.4 Thesis Outline

Chapter 2 discusses the problems of Mobile Ad Hoc Networks. In relation with MANETs, two emerging high rate wireless data communication technologies are discussed

Chapter 3 reviews the related research for this work. This includes an in-depth discussion on existing collaborative frameworks and argues why existing support for building collaborative applications in Mobile Ad Hoc Networks is not sufficient.

In chapter 4 the architecture of the framework is presented.

Chapter 5 presents the TransPeer system, an implementation of the conceptual framework described in Chapter 4. The system not only contains this implementation, but also includes two samples to show the strength of the framework.

Finally, Chapter 6 contains a summary and conclusion with suggestions for future research.

1.5 Summary

This chapter started with an introduction to PDAs. While PDAs are currently used for simple tasks like maintaining a calendar, the possibilities will soon be there to move the PDA towards a ubiquitous computing environment. Mobile Ad Hoc Networks and usage were discussed. Since PDAs are mobile devices without pre-defined access to a network, they can't rely on an existing infrastructure. Two examples were given to show the relation between Mobile Ad Hoc Networks and PDAs. Section 1.2 discussed the need for a framework to support collaboration in Mobile Ad Hoc Networks and gave a definition. Finally, section 1.3 gave the research question of this thesis and the contribution to the research in Mobile Ad Hoc Networks.

CHAPTER 2

MOBILE AD HOC NETWORKS

Amateur radio, also known as ham radio, provides wireless communication in frequencies between 1.8Mhz and several hundred gigahertz. It received most attention at the beginning of the 80's, but is still used by over a million hobbyists around the world. Packet radio is a particular digital mode of amateur radio, which splits up data into packets that can be sent and received by stations called TNCs (terminal node controller). Since the beginning of the TNC standard, researchers have tried to address routing problems. Stations were turned on and off frequently, resulting in routing failures. In the early days of packet radio, repeaters (also known as digipeaters) were used to allow a greater range. A packet was simply repeated until it had reached its destination. This scheme worked as long as the number of users was small. However, as packet radio became more popular, digipeaters were soon clogging up the network with traffic being repeated over long distances. Mobile stations formed yet another challenge on how to route packets. New standards were required and did emerge, eventually even resulting in the option to use TCP/IP over packet radio.

Packet radio lost the attention of researchers due to the Internet, which provided a much higher transmission rate. However, problems that have been identified already 20 years ago need to be solved again for mobile devices.

Advances in wireless technology and portable computing along with demands for greater user mobility have provided a major thrust towards development of an emerging class of self-organizing, rapidly deployable network architectures referred to as Mobile Ad Hoc Networks [1][8]. A Mobile Ad Hoc Network (MANET) requires no fixed infrastructure. Any device with a microprocessor, whether mobile or stationary, is a potential node in an ad hoc network. This includes mobile telephones, motor vehicles, roadside information stations, satellites, and desktop or hand-held computing devices. As mentioned before, MANETs cannot rely on routers for path discovery and traffic routing. Consequently, mobile end-systems in a MANET are expected to act cooperatively to route traffic and adapt the network to the highly dynamic state of its links and its mobility patterns.

The following figure (figure 1) shows three nodes that are connected in an ad hoc mode where, A can see B and C, whereas B and C can only see A. Node A may potentially route communication between B and C.

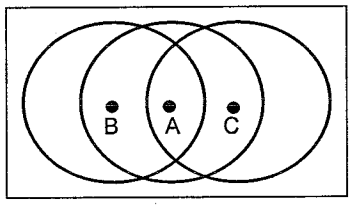

Figure 1: An ad hoc network topology containing the nodes A, B and C.

The following list summarizes the characteristics of MANETs:

- Spontaneous network creation, without (necessarily) using a pre-existing infrastructure.
- Formed by wireless hosts, which may be mobile.
- Hosts can join and leave the network at any time.
- Routes between nodes may potentially contain multiple hops.

There is a widespread interest in MANETS due to their well-known advantages for certain types of applications. Since a fixed infrastructure is not a prerequisite, a wireless ad hoc network can be deployed quickly. This makes these networks suitable to situations where either there is no other communication infrastructure present or where such infrastructure cannot be used because of security, cost, or safety reasons. New terminals can be added easily to the network and individuals can move easily in the network in order to perform their primary functions without a concern for maintaining communications with other entities.

2.1 Challenges of Mobile Ad Hoc Networks

The organization and control of MANETs poses some challenges not seen in traditional networks.

- Since the topology of the network is constantly changing, the primary challenge is routing with frequent link failure and disconnections. The distribution of information can easily saturate a network; late arrival of information can drive a network into instability.
- Multicast routing will be a challenge because the multicast tree is no longer static.
- Providing different quality of service levels (QoS) in a constantly changing environment will be a challenge. An adaptive QoS must be implemented on top of the traditional resource reservation.
- Conservation of power and power-aware routing must be taken into consideration, because mobile devices typically have a limited power source. Routing may be desirable on more "long-lived" routes.
- The need for location-aided routing. Using positioning information could lead to a less time-consuming path discovery.

Until recently ad hoc networks were mostly used for military applications. However, due to the advantages mentioned above, one can expect many applications to emerge in the commercial markets. The establishment of the Internet Engineering Task Force, Mobile Ad Hoc Networks Working group [5], which works on IP routing in ad hoc networks, shows that there is a widespread interest in these networks in the scientific community as well as the industry.

The framework presented in this thesis will not address any of the challenges of ad hoc networks. Since the MANET Working Group is continuously working on a set of protocols, it was decided not to address these issues but instead identify the problems and take them into account in the design and development of the framework. Chapter 3 will present a flexible framework that allows adding future implementations of network protocols with small effort. Instead of using non-standardized protocols, the framework currently uses the well-known TCP/IP stack as a communication protocol. Although this protocol set is not really suitable for mobile networks, it was decided not to experiment with protocols that are not standardized yet.

2.2 Data Communication Technologies

To enable collaboration between applications in Mobile Ad Hoc Networks, data communication (datacom) technologies like W-LAN (IEEE 802.11b) and Bluetooth are rapidly being deployed. These technologies provide low range (typically around 10 meters) connectivity. Such datacom networks will soon populate strategic places, including airports and city centers. These "hotspots" cannot only provide the users with local infotainment but could also be used as gateways to the Internet.

Figure 2 shows a roadmap for PDAs and the availability/adoption of these technologies in various Operating Systems [7]. It is out of the scope of this thesis to present an overview of existing PDAs and their Operating Systems. However it must be noted here that there is a wide variety of PDAs and Operating Systems available today, each with a reasonable market share. Below only the most used are depicted.

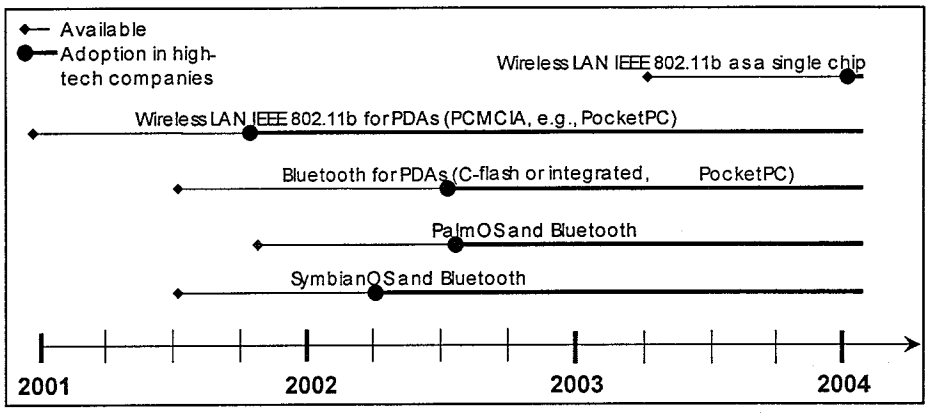

Figure 2: Roadmap for PDAs with ad hoc networking capabilities

Below, two promising wireless datacom technologies are introduced.

2.2.1 Bluetooth

Bluetooth [3] is named after the 10th Century Danish King Harald Blåtand and is a considered to be very promising among wireless developers. Bluetooth chips were originally meant to replace the cables between a printer and desktop for instance, but soon it became clear that Bluetooth is capable of much more than just replacing cables. Today, Bluetooth is seen as a general wireless communication layer between any Bluetooth enabled device. Bluetooth operates in the 2.4 GHz unlicensed Radio Frequency (RF). Currently it can connect devices that are up to ten meter apart and it offers a total bandwidth of 1Mbit/sec. Future specifications will offer a wider range and more bandwidth.

The Bluetooth specification is an open specification that is governed by the Bluetooth Special Interest Group (SIG). The Bluetooth SIG is lead by its five founding companies and four new member companies who were added in late 1999. These nine companies form the *Promoter Group* of the Bluetooth SIG:

Founding Companies	New Members
Ericsson	3Com Corporation
IBM Corporation	Lucent Technologies
Intel Corporation	Microsoft Corporation
Nokia	Motorola Inc.
Toshiba Corporation	

Table 1: Members Bluetooth SIG

More than 1200 additional companies are members of the Bluetooth SIG. The magnitude of industry involvement should ensure that Bluetooth becomes a widely adopted technology

Bluetooth devices can interact with one or more other Bluetooth devices in several different ways. The simplest scheme is when only two devices are involved. One of the devices acts as the master and the other as a slave. This ad hoc network is referred to as a "piconet". A piconet is any such Bluetooth network with one master and one or more slaves. In the case of multiple slaves, the communication topology is referred to as point-to-multipoint. In this case, the bandwidth is shared among all the devices in the piconet. There can be up to seven active slaves in a piconet. There can be additional slaves, but do not have an active address. These slaves are not active and are referred to as parked.

When two piconets have an overlapping coverage area, the Bluetooth specification refers to it as a scatternet. Slaves in one piconet can participate in another piconet as either a master or slave. Figure 3 shows an example of two overlapping piconets.

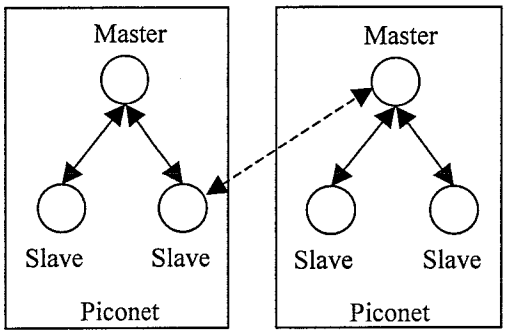

Figure 3: Overlapping coverage, a "Scatternet"

2.2.3 Wireless LAN (IEEE 802.11b)

IEEE 802.11b "High Rate" standard wireless local area network (WLAN) also operates in the 2.4GHz (2.4 to 2.483 GHz) unlicensed Radio Frequency (RF) band and can transmit up to 11Mbps (Megabits per second). It was released in September 1999 as addition to IEEE 802.11.

IEEE 802.11b defines two types of equipment, a wireless client with a wireless network interface card (NIC), and an Access Point (AP) that acts as a bridge between the wireless stations and wired networks. There are two operation modes in IEEE 802.11b, Infrastructure Mode and Ac Hoc Mode.

1. *Infrastructure Mode*
   Infrastructure Mode consists of at least one Access Point connected to the Distribution System. This mode has two different configurations:
   - *Basic Service Set (BSS)*
     An Access Point provides a local bridge function for the BSS. All wireless stations communicate with the Access Point and no longer communicate directly. All communication is relayed between wireless stations by the Access Point.
   - *Extended Service Set (ESS)*
     An Extended Service Set is a set of infrastructure BSS's, where the Access Points communicate amongst themselves to forward traffic from one BSS to another to facilitate movement of wireless stations between BSS's.

2. *Ad Hoc Mode*
   Independent Basic Service Set (IBSS) or Peer-to-Peer
   The wireless stations communicate directly with each other. Every station may not be able to communicate with every other station due to the range.

Figure 4 depicts the two different operation modes in IEEE 802.11b.

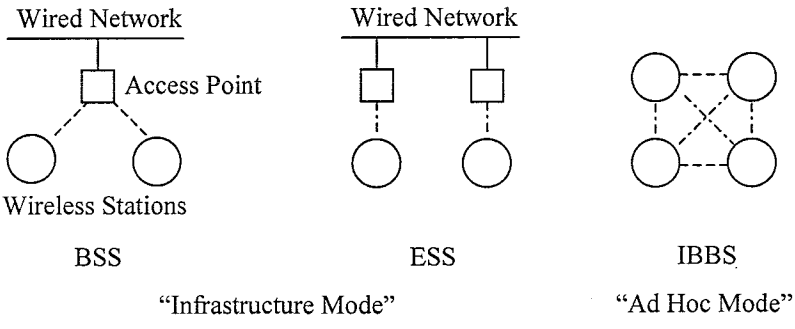

Figure 4: Operation modes in IEEE 802.11b

The difference between the infrastructure mode and the ad hoc mode is the use of access point(s). In the ad hoc mode a network can be formed without the presence of an access point

2.3 Summary

This chapter started with a short history about amateur radio. Already then, routing problems formed a great challenge to overcome for researchers. The same problems identified back then recently found attention again. MANETs and two communication technologies in relation with MANETs were discussed.

CHAPTER 3

RELATED WORK

This chapter will discuss three initiatives similar to the one that will be presented in this thesis. It must be noted that there are more of these initiatives well under way, mostly coming from the academic world. However, only three well-known architectures coming from the industry are discussed here.

Part of this thesis is to relate the framework presented to the current available collaboration frameworks. There is little benefit of developing yet another framework with similar functionality as offered by current frameworks. This chapter will first introduce some properties relevant for a discussion. The following sections will discuss how the frameworks meet the various properties. The last section will provide a motivation for a new framework that will be discussed in this thesis.

3.1 Relevant Properties

With the ubiquitous computing area evolving, collaboration between devices has become a serious research issue. A number of frameworks addressing mobile and specialized devices have emerged recently. Essentially, these are coordination frameworks that propose certain ways and means of device interaction with the aim of device inter-operability [22].

Device coordination essentially means providing a device with a subset of the following capabilities:

- *The ability to announce its presence to the network.*
  Devices must be able to announce their presence before they can be part of the network.

- *Discovery of devices in the neighborhood.*
  Apart from announcing its own presence to the network, a device must be able to discover other devices that are part of the network.

- *Ability to describe its capabilities as well as query/understand the capabilities of other devices.*
  Discovery of a device only reveals its existence, it does not describe what kind of service the discovered device offers. Therefore, a device must be able to describe its capabilities and furthermore understand the capabilities of others.

- *Invocation of services*
  After finding the requested service, it must be usable.

- *Self-configuration without administrative intervention.*
  As described in chapter 2, MANETs are highly dynamic. Self-configuration means that the framework will automatically adapt any client to its changing environment.

- *Inter-operability with other devices.*
  The diversity of networked devices is growing rapidly, all with their own properties (e.g. operating system). These devices would benefit from a framework that is able to support communication between two different devices.

Apart from these coordination requirements, the following issues are identified as relevant for the discussion:

- *Security of the framework*
  Any device that connects to a network has a risk of being used by unauthorized parties.

- *Network infrastructure transparency*
  There exist a wide variety of network infrastructures (e.g. infrared, Bluetooth, Wireless LAN). Devices would benefit from a framework that has the potential to communicate over different network infrastructures.

Both the description of a service, as well as the invocation of a service requires some standardization. For instance, a printer might advertise itself as "print", referring to the service is offers, but instead could also use the word "printer", referring to the type of device. Clearly, the semantics of describing a service requires standardization. Furthermore, after the printer is identified, how can a device actually print something? Again, this requires knowledge about the methods exposed by the service and can only be accomplished by standardizing these exposed methods.

For any coordination framework to work, it must introduce some standards into the operations of the framework. Otherwise the framework simply cannot coordinate. The essential problem here is maintaining a balance between standardization requirements and autonomy. Here, autonomy means the use of proprietary protocols and techniques and the need (market-based or otherwise) to continue using them.

The following sections will identify the frameworks by describing how they address the above-mentioned issues.

3.1 Jini

Jini from Sun Microsystems [28] is tailored specifically to Java. The developers of Jini were inspired by Linda [4] coordination model using Tuple Spaces, developed about 15 years ago. This resulted in the JavaSpaces [27] technology with its later evolution into Jini.

Jini uses the term federation to refer to coordination between devices. A federation is a collection of services offered by devices that can become aware of one another and collaborate if needed. To facilitate this, a Jini system contains a number of lookup services that maintain information about available devices and their offered services.

The Jini lookup service is the central part of a Jini system. Every device must discover such a lookup service before it can enter a federation. Thus the registration process of a Jini system follows the typical client/server model, where the devices act as clients locating the central lookup service. The location of a lookup service may be known before hand, or it may be discovered using a multicast. Once a device has located a lookup service it can register with it. During registration, it can publish a set of properties (which are name/value pairs). A lookup service matches queries from other devices against the published properties.

During the registration process it is possible for a device to upload some Java code to the lookup service. This code is essentially a "proxy" that can be used by other devices to contact an interface on the provided service in question. This is accomplished using Remote Method Invocation (RMI) in Java that allows a Java program to call a method in a remotely running Java program through some exposed interface. Both devices must have Java RMI embedded for this to work.

An advantage of Jini is its portability. Since it is entirely written in Java, it can rely on the "run-anywhere" property of Java code.

Table 2 presents a summary of how Jini addresses the various relevant properties

Property	Support
Announcing Presence to the network	Registration with a Jini lookup service
Discovering other services	Devices can query a lookup service for services that meet properties of interest.
Describing capabilities	Registration information in the lookup service contains properties.
Network configuration	Not addressed. The use of the Java platform implies it cannot directly address native operating system properties.
Invoking services	Invocation is provided through Java RMI, which requires standardization of exposed methods.
Inter-operability with other devices	Jini requires the Java platform to be present on the device.
Security	Not more than the basic security offered by Java and RMI, which provides ways to grant/deny access to resources.
Network transparency	Relies on TCP/IP stack, thus not transparent.

Table 2: Summary of Jini

3.2 UPnP

UPnP [18] is a peer-to-peer network architecture allowing automatic advertisement, discovery and control of devices over the network without user intervention and leveraging existing open standards such as IP suites and HTTP. Though supported by Microsoft, it intends to be able to run on any operating system and platform. Although it is considered at some levels to be a natural extension of Microsoft Plug and Play [17] to the networking scenario, its implementation framework has no resemblance to it.

While Jini can rely on a "run-anywhere" solution with the use of Java code, UPnP uses a protocol suite that is implemented almost anywhere: the TCP/IP stack. UPnP works primarily with these lower layer network protocols, proposing new standards at this level instead of at the application level. This primarily involves additions to the IP suite, in the form of certain optional protocols that can be implemented natively by devices. UPnP attempts to make sure that all device manufacturers can quickly comply with the proposed standards without major hassles.

Devices use a protocol known as Simple Service Discovery Protocol (SSDP) [31] to announce their presence to the network and can use it to discover other devices. A URL, pointing to an XML file that describes the capabilities of the service, accompanies the announcement. A lookup service[1] can be used to announce the presence, but it can also use multicasts when no lookup service is available. Furthermore, the lookup service can be queried to find a particular service. If no lookup service is available, a service can multicast a query. Any listening service that matches the query can then respond. Again, standardization is required to agree upon the semantics of how to advertise a particular offered service. The ability to invoke a service is not present in UPnP; it focuses on device discovery and describing capabilities.

Unlike Jini, UPnP does address the problem of automatic assignment of IP addresses and DNS names to a device being plugged in. To accomplish this, some additional protocols have been proposed. The devices first look for a DHCP server [6] on the network. If no DHCP server is present, the device resolves a free IP by using the ARP protocol [21] and assigns it to itself. This is known as AutoIP. To address issues in naming, a multicast DNS proposal has been drafted. This would enable plugged-in devices to both discover DNS servers via multicast and also resolve DNS queries about itself via multicast.

[1] In UPnP, a lookup service is called "proxy". However, the word "proxy" is more commonly used in a different context and therefore "lookup service" is used.

Property	Support
Announcing Presence to the network	A device uses SSDP to announce its presence.
Discovering other services	A device can both listen to the SSDP multicast channel directly or can contact a lookup service.
Describing capabilities	An XML file describing the service offered by the device.
Network configuration	DHCP or AutoIP is used to configure the network.
Invoking services	Not addressed.
Inter-operability with other devices	Potentially, after standardization and adoption of SSDP, it can reach a wide variety of devices.
Security	Not addressed.
Network transparency	Relies on TCP/IP stack.

Table 3: Summary of UPnP

3.3 Salutation

The Salutation Consortium's architecture starts from the position that network architectures should be free of vendor-imposed limitations [20]. Salutation is not limited to nor does it have a prerequisite for Java, UDP or HTTP. It is platform, OS, and network independent. The Salutation Architecture does not assume a single pervasive infrastructure. The strength of this architecture lies in its ability to support multiple infrastructures through a single implementation (figure 5)

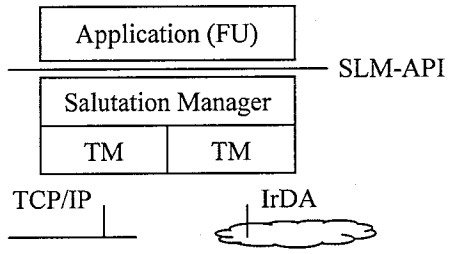

Figure 5: Layers Salution

Applications can talk directly to a Salutation Manager (SLM), which may be in the same device or located remotely. To explore the environment, a Salutation Manager can team up with other Salutation Managers. The SLM can be seen as an agent that does everything on behalf of their clients. Data transfer between applications, including those across different media and transports, is mediated through them.

An application must register with an SLM. The SLM in turn will exchange registration information with other SLMs, even with those on different transport media. This is done through the Transport Manager (TM), which is a transport-dependent module. A Functional Unit (service), representing some essential feature (e.g. Print, Scan, etc.), has a collection of attribute pairs (name, value) to describe its features. These pairs can be queried and matched against during the service discovery process. A discovery request is sent to the local SLM which in turn will be directed to other SLMs. SLMs talk to one another using Sun's ONC RPC (Remote Procedure Call) [30]. Salutation defines APIs for services to invoke these operations and gather the results.

Once the requested service has been found, Salutation uses "drivers" (comparable with Jini's proxy) to invoke the service. The driver might be available on the requester's side, but can also be downloaded from the network. Again, standardization is needed of the exposed methods of a service.

Property	Support
Announcing Presence to the network	A device registers with a Salutation Manager.
Discovering other services	A device can query a SLM.
Describing capabilities	Each device registers with a collection of attribute pairs (name, value) to describe the service it provides.

Network configuration	Not addressed.
Invoking services	The Salutation Consortium is working on standardization of interfaces of different Functional Units.
Inter-operability with other devices	A large variety of devices is already supported.
Security	Salutation offers user authentication with a user-id and password scheme.
Network transparency	The key feature of Salutation: its transport independent architecture.

Table 4: Summary of Salutation

3.4 Overview

This chapter discussed three different emerging collaboration frameworks all with their strengths and weaknesses. Jini provides a standardized way to contact services through the use of Java RMI. However, this implies Jini depends on Java RMI in order to work. Currently support for Java RMI on PDAs is limited or not present at all. The strength of UPnP is its network configuration. None of the other discussed architectures addresses this issue (including the one presented in this thesis). But it should be mentioned that these features of UPnP are addressed in the next generation Internet Protocol (IPv6) [9] and hence should be available to the other frameworks in the future. Salutation seems to have a unique key feature with its OS and network independence.

However, the framework presented in this thesis differentiates from the ones described above due to the envisioned settings in which it will be used:

- *One type of device*
  While the above-mentioned frameworks focus mainly on interoperability between different types of devices (e.g. printer, fax), the framework presented in this thesis focuses solely on use within PDAs, which lowers the overhead. This resulted in the decision that no effort is put in standardization of describing capabilities. A short description of a service will be presented to the user upon discovery. The user is then in command to choose to use it or not.

- *Mobile Ad Hoc Networks*
  The framework is targeted at Mobile Ad Hoc Networks. The client/server model used in Jini and partly in UPnP can no longer be used without the existence of a centralized server.

- *Network independent*
  PDAs can already communicate through various wireless network protocols (e.g. Bluetooth, TCP/IP, IrDA). While currently fixed infrastructure networks mostly use TCP/IP, the battle in the wireless world is long before over. Whichever is available on the PDAs, the framework should be able to use it. Salutation offers network independence, but Jini and UPnP both highly depend on the TCP/IP stack.

- *Sharing, rather than invoking*
  Both Jini and Salutation offer invocation of services. However, they both suffer from standardization issues. They both essentially are trying to trade proprietary drivers for universal standards, something that is not very easy to realize. Instead, the framework presented in this thesis allows services to be shared (downloaded) within the network. Say a user discovers a chat service. He/she can then choose to download this chat service and start to use it. A service can be implemented with its proprietary communication protocol. Of course this poses some serious security issues to be solved (see chapter 6.2)

The issues mentioned above were reason enough to design and develop a new collaboration framework, rather than extending or modifying an existing framework.

CHAPTER 4

FRAMEWORK FOR COLLABORATION IN AD HOC NETWORKS

This chapter describes the architecture of the framework in detail. Section one introduces the context of the framework, what it should provide and why. Then in section two the design principles are discussed. The architecture of the framework is presented afterwards, using the "4+1" model [15]. Section eight will discuss how the requirements from section two are met.

4.1 Context

Chapter one stated the need for a framework for collaboration. Also some examples were given that showed the need for some sort of software to enable the communication between clients. Mobile Ad hoc networks were discussed and their relation with PDAs. As mentioned, PDAs will be used more and more in mobile environments. Figure 6 depicts the environment of a user and his PDA.

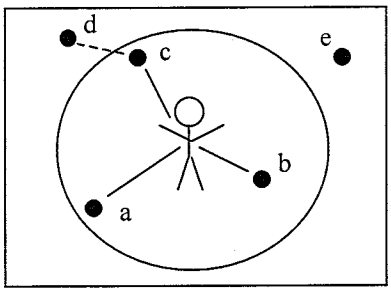

Figure 6: Target area of framework

The user is depicted in the middle. The circle represents the range of the network being used by the user (e.g. Bluetooth). The black dots represent peers that are "visible" for the user. A visible peer is a peer that is able to communicate with the user, i.e. that is part of the network used by the user. Proximity, in relation with the user, can then be defined as all visible peers. Peers *a*, *b* and *c* are in proximity of the user, whereas peers *d* and *e* are not. Peer *c* might act as a router to establish a connection between peer *d* and the user, however routing is not supported in the framework today. It is possible that humans control these peers, but as stated in the museum example (chapter 1) it could also be a standalone computer.

Different peers can provide different services to the user. While one peer might represent an electronic ticket booth at a train station, offering a last-minute ticket, another peer might be a colleague sending the latest notes of a meeting to the user using the conference suite described in chapter 2.

The framework must enable the user to explore his/her proximity and discover which services are available in proximity. Furthermore, it must enable the user to use all services available on peers within proximity.

The next section will formulate the design principles and requirements for the architecture

4.2 Design Principles and Requirements

This section will present the design principles and requirements of the framework. During the thesis some of these have been added or eliminated due to changes in the envisioned settings of PDAs. The following principles and requirements are seen as stable and lead to a basic framework.

In order to state the principles and requirements, first the system boundaries must be defined. The system consists of:

- The various PDAs/PCs (other clients) equipped with the framework.
- The infrastructure to communicate

PDAs and their capabilities are rapidly developing. Therefore, the framework shall be able to adapt to future functionalities with ease. Although the capabilities of PDAs are growing, they are still very limited in resources (e.g. memory). These issues must be taken in consideration for the development of the framework:

1. *The framework shall be targeted at, but not limited to, PDAs.*
   PDAs are limited in resources. Memory capacity is low and computational power is more of an issue than on desktop computers. This implies care must be taken to keep the framework as small and simple as possible.

2. *The framework shall be designed in a way that tolerates the evolution of PDAs and their capabilities.*
   The PDA revolution is just starting. The coming years will show a wide variety of devices all with their own characteristics (hardware, OS, etc). For example, a GPS system will soon be available for PDAs. This new functionality must be easily added to the framework.

The framework shall enable discovery and usage of services in proximity. Because of the unpredictable nature of the targeted environment, peer-to-peer communication shall be used, since peer-to-peer applications do not rely on centralized servers. Various types of wireless networks are available and currently there is no overall "winner" among these technologies. Therefore, the framework shall be able to support various types of wireless networks (e.g. Bluetooth, WLAN, IrDA, etc). The development of services will be the driving force behind the maturity and acceptance of the framework. To aid the development of services, the framework should be usable in a straightforward way. This leads to the following principles and requirements:

3. *The framework shall use a peer-to-peer communication model to collaborate with its environment.*
   Without the existence of centralized servers (ad hoc networks), communication between PDAs shall be in a peer-to-peer fashion.

4. *The framework shall be able to support various types of wireless networks.*

5. *The framework shall be straightforwardly usable by service developers.*

To enable communication between PDAs, they shall be able to announce their presence to the network. Furthermore, after announcement the PDAs shall be able to discover the services (and PDAs) in proximity. After discovery the services should be able to communicate with each other.

6. *The framework shall enable the discovery of devices and their services in proximity.*
   This includes the ability to describe its capabilities as well as query/understand the capabilities of other services.

7. *The framework shall enable the use of services, both local and in proximity*
   To enable the use of services, execution code might be needed. This means the execution code shall be made available over the network.

4.3 Logical View

The "4+1" view model suggest that the logical architecture should primarily support the functional requirements, what the system should provide in terms of services to its users [15].

4.3.1 Top-level Logical View

Below, in figure 7, a top-level logical view of the framework is presented.

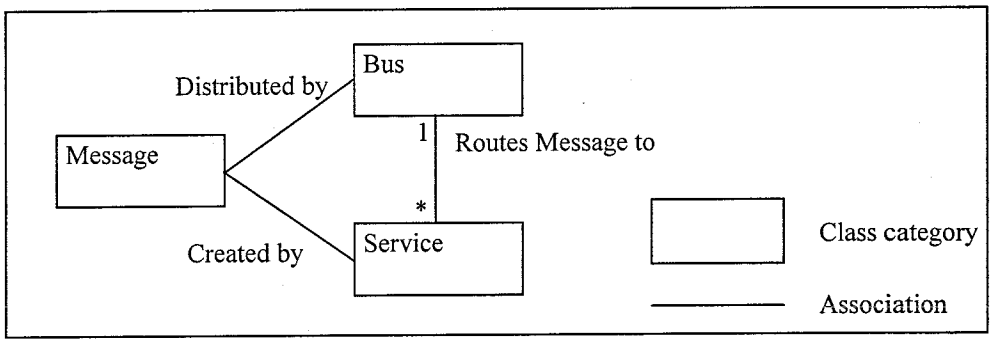

Figure 7: Blueprint framework

The above figure applies to one PDA. A functionality (or group of functionalities) is captured in a *Service*. A *Service* can send and receive a *Message*. A *Message* is targeted to a specific *Service*, has a command (to aid the receiving side), some data, and an originator (the service sending the message). The *Bus* acts as a coordinator between local services, passing the messages to the right services. Every *Service* must register with the *Bus* in order to send and receive a *Message*.

4.3.2 Detailed Logical View

The required functionality of the framework is provided by several services. The framework consists of core services that provide the required functionality, and additional services. Although a user can only interact with the framework through a Graphical User Interface (GUI), the framework can also run on a standalone computer that does not need a GUI. Therefore, the GUI is an additional service.

The following table summarizes which functionality is captured in which service:

Functionality	Provided by	Core/Additional
Discovery of devices	NetCenter	Core
Sending/receiving data to/from other devices	NetCenter	Core
Download/upload of services	Explorer	Core
Start/Stop services	Explorer	Core
Event mechanism	EventCenter	Core
Presentation of available services/devices	GUI	Additional

Table 5: Services and their functionalities

NetCenter provides the discovery of peers. Once a peer is discovered, *NetCenter* can be used by local services to communicate with the remote services. *Explorer* is used to start and stop local services. Furthermore, it is responsible for the transfer of a service (both upload and download). *EventCenter* provides an event mechanism for local services. A service can register with *EventCenter* and state the events it wants to be notified of. For example, when a new device is detected in proximity, *NetCenter* sends an event message to *EventCenter*. All services that registered with *EventCenter* for this particular event will be notified. *GUI* is an additional service that presents the available devices and services in proximity.

Figure 8 shows the logical view of the framework.

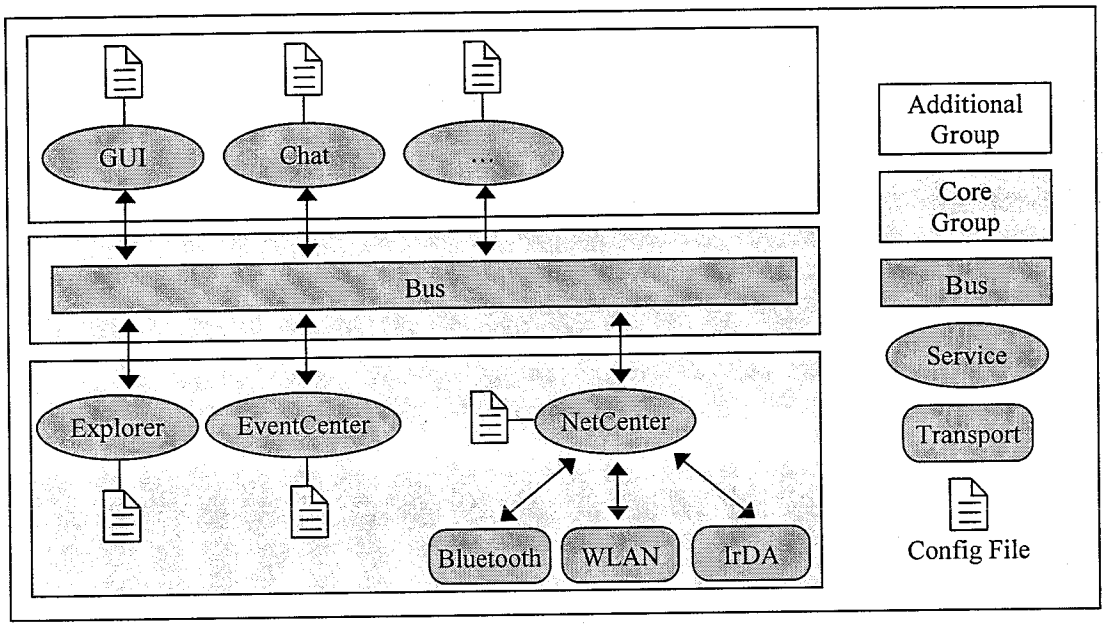

Figure 8: Logical view of core framework

A service can send a *Message* to another service. The *Bus* routes the message to the right service. An example *Message* is presented in figure 9.

```
Message
{
ToService: "NetCenter"
FromService:  "Chat"
Command: 0
Data:
<NetMessage>
   <Transport>Bluetooth</Transport>
   <ToPeer>Robin</ToPeer>
   <RemoteService>Chat</RemoteService>
   <RemoteCommand>0</RemoteCommand>
   <Data>Hello Robin!</Data>
</NetMessage>
}
```

Figure 9: Example Chat Message

The *Chat* service constructs a *Message* object by specifying the local receiver (*NetCenter*), the sender (*Chat*), a command, and some data. The *Bus* will dispatch the *Message* to the local service *NetCenter*. *NetCenter* will check the command, in this case 0, which is interpreted as a *Message* to be sent over the network. Because this *Message* should be sent over the network, *NetCenter* will parse the data to determine the peer to which this *Message* should be sent. Additionally, the transport type to be used, the remote receiving service, the remote command and some data, are determined. Then the object is serialized and sent to the receiver's (in this case Robin's) *NetCenter*. The remote *NetCenter* will construct a *Message* object and pass it to *Bus*, which in its turn will pass it to the *Chat* service.

As shown, *NetCenter* provides services with various transports (e.g. Bluetooth). While the OSI [33] reference model suggests abstraction of the transport used, it was decided to actually do provide services with the option to choose a particular transport. Reasons for this decision were, QoS, transfer rate and battery consumption. While WLAN (IEEE 802.11b) can potentially provide high transfer rates

it cannot guaranty any lower bound for this transfer rate. An FTP like service would typically use WLAN. Bluetooth, however, can guarantee QoS. An example service requiring Bluetooth would be a multi-user network game. In order to keep the users satisfied (e.g. absence of network congestions), Bluetooth can be used to guarantee a lower bound for the transfer rates. IrDA consumes less battery power while still providing reasonable transfer rates. It can typically be used for less intensive services such as a chat. Instead of explicitly stating a particular transport, a priority model could be used by *NetCenter*. Services could state a priority level for the transport to be used. However, it was decided to put the responsibility of choosing a transport not in *NetCenter*, but instead let the particular service choose.

Every service has a configuration file specifying various parameters needed by the framework. Figure 10 presents an example configuration file for the *Chat* service.

```
<Config>
  <Name>Chat</Name>
  <Description>Talk with your friends in proximity!</Description>
  <Files>
    <File>Chat.class</File>
    <File>ChatGUI.class</File>
    <File>config.xml</File>
  </Files>
  <ConfigFile>config.xml</ConfigFile>
  <Platform>Java 1.1.8</Platform>
  <Execute>
    <Command>java</Command>
    <Argument>Chat</Argument>
  </Execute>
</Config>
```

Figure 10: Example configuration file

The *Name* field specifies the name of the service. The *Bus* will use this name as a reference to the service. To provide remote peers with some information about the service, a *Description* field is used. The *Explorer* uses the *File* and the *ConfigFile* fields when uploading a particular service to a remote peer. The *Explorer* also uses the *Platform* and *Execute* fields to determine how to start the service.

XML [32] is used both by the configuration file as by the data field of a *Message* object. It was decided to use XML since it is widely accepted and available on almost every platform.

4.4 Process View

The process view for the framework is depicted in figure 11.

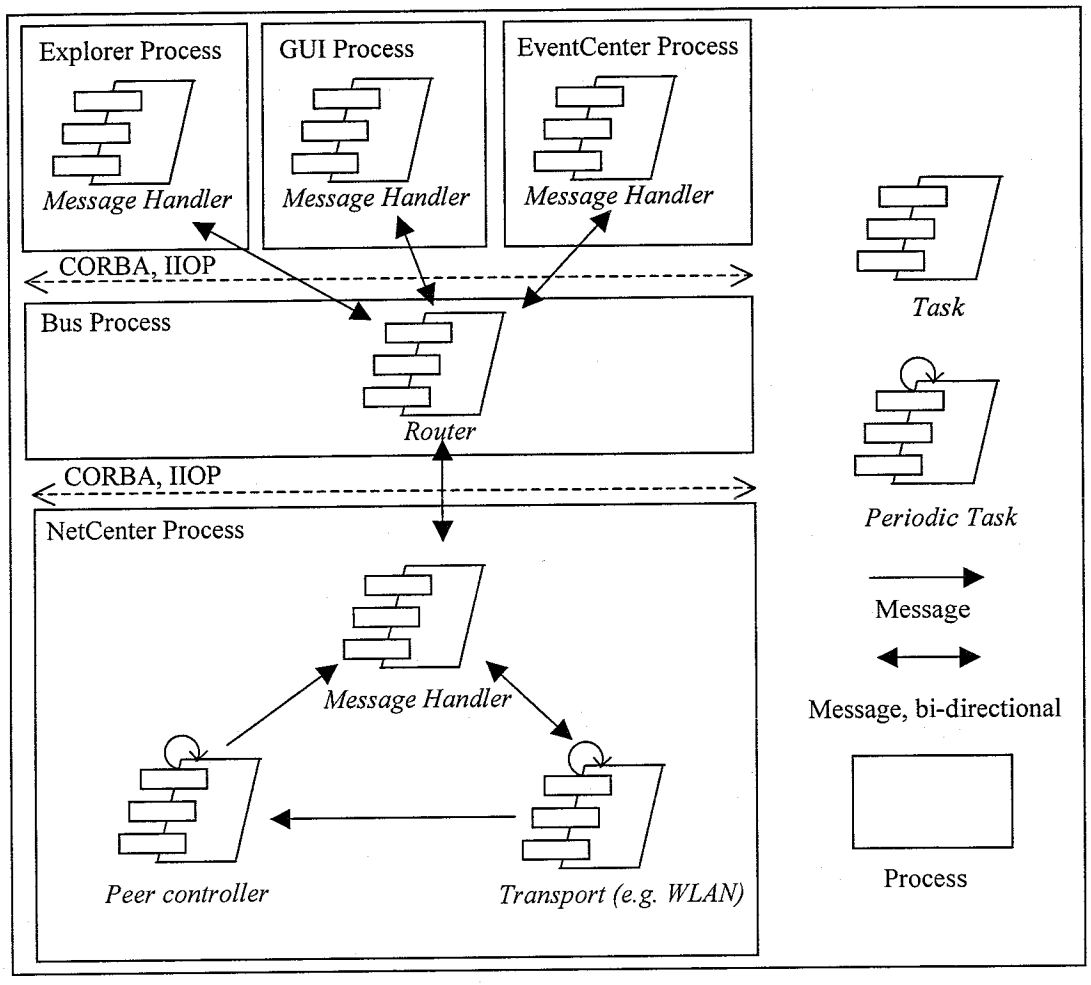

Figure 11: Process view of framework

The *Bus* and all services (e.g. *NetCenter*) are separate executable units. To support services being written in different languages (e.g. C++, Java) it was decided to use CORBA [19] as middleware between the different processes. The CORBA implementation used by the framework creates a new thread of the server object (*Bus*) for each client. As a consequence the *Bus* process will have *n* threads running when *n* services are connected to it. Access to shared data within the *Bus*, for example the references to the different services, must be synchronized among the threads. Chapter 5 will show how access to this shared data is synchronized.

The *Router* task of the *Bus* process routes the incoming messages to the right services. All services have a *Message Handler* task that is responsible for handling both messages from the *Bus* as well as messages to the *Bus*. The *Message Handler* task of *NetCenter* is driven by both incoming messages from the network (passed by a *Transport* task) as well as messages that should be sent over the network (passed by Bus). The various transports (e.g. WLAN, Bluetooth, IrDA) periodically check for other devices in proximity. Different transports provide different ways to accomplish this. For instance Bluetooth and IrDA provide developers with an "inquire" method that returns all the Bluetooth/IrDA enabled devices in proximity. WLAN is using TCP/IP, which does not provide such methods. Instead, the WLAN transport periodically multicasts "alive" messages to announce its presence on the network. The *Peer Controller* task periodically updates the peer list (i.e. the peers that are part of the network) when "alive" messages and results from the "inquire" methods are passed by the various transports. Peers that appear to be offline for over a period of time are deleted from the list.

4.5 Development View

The development view focuses on the actual software module organization on the software development environment. The software is packaged in small chunks (libraries, or subsystems) that can be developed by one or more small number of developers. The subsystems are organized in a hierarchy of layers, each layer providing a narrow and well-defined interface to the layers above it [15].

Figure 12 depicts the development organization of the framework in five different layers.

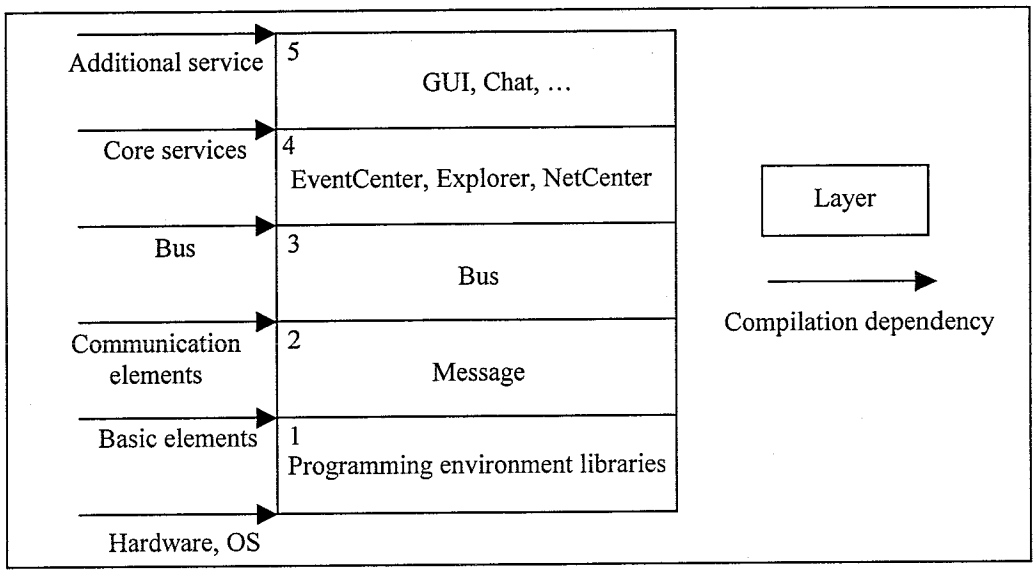

Figure 12: Development view of framework

Layer 1 contains the basic building blocks of the framework. The *Basic elements* layer contains the libraries provided by the programming environment. The *Communication elements* layer contains the elements that are passed between the different services and the bus (e.g. *Message*). Layer 3 and 4 form the core group (see figure 8) of the framework. The *Additional service* layer contains all the services not critical for the framework to execute properly and depend on the core services provided by the framework.

4.6 Physical View

The physical view maps the software to the hardware. Figure 13 shows three PDAs, all equipped with the framework. Communication between the PDAs is provided by the various available transports (see figure 8) of the *NetCenter* process. The *Bus* process routes the different incoming messages (from *NetCenter*) to the intended receiver.

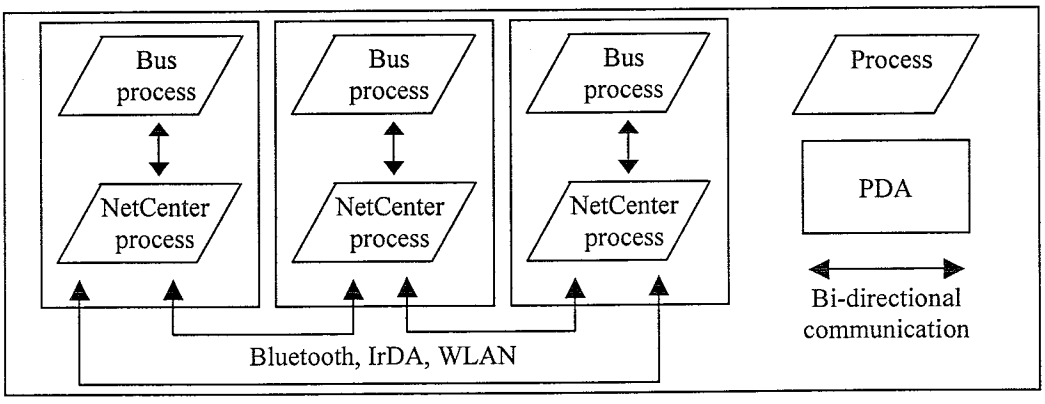

Figure 13: Physical view of framework

4.7 Scenarios

The elements in the four presented views are shown to work together seamlessly through the use of a small set of important scenarios [15]. UML sequence diagrams [23] are used to present the various scenarios. The following scenarios will be presented in this section:

- Starting a service
- Peer discovered
- Peer "offline"
- Exploring the services offered by a peer
- Downloading a service from a peer

It must be noted that not all method-parameters are shown in the various diagrams.

4.7.1 Starting a Service

This subsection will present the sequences of interactions between the various services when a new service is started. Figure 14 presents a high-level sequence diagram of this scenario. In this case, the request to start a service comes from *GUI*. However, the request to start a service can come from any service. *GUI* constructs a message, ordering *Explorer* to start a particular service, and passes it to *Bus*. *Explorer* will get the command line arguments needed to start the service. Once *Chat* is started it will register with *Bus* to become part of the framework.

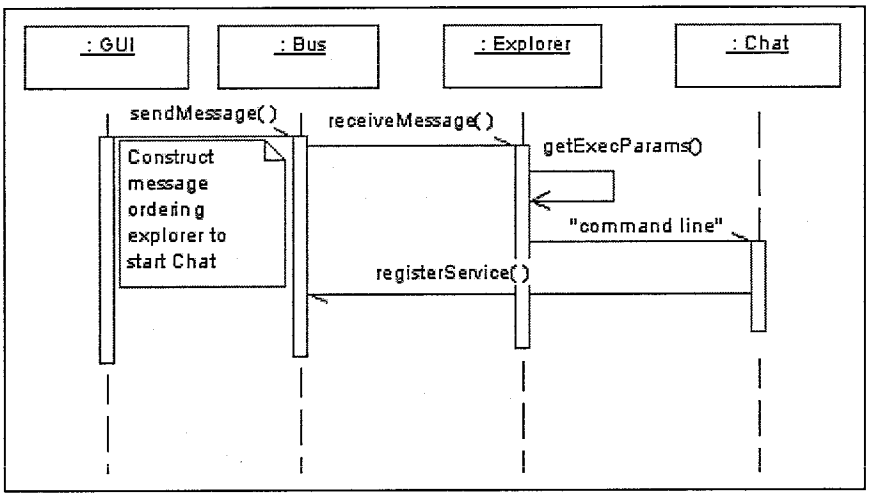

Figure 14: Starting a service

4.7.2 Peer Discovered

Figure 15 shows the sequences of interaction when a peer is discovered via the WLAN transport. A remote device periodically multicasts so called "alive" messages. A receiver of such a message will check if it involves a new peer. If so, *NetCenter* sends an event message containing the peer's name and the transport that discovered this new peer. *EventCenter* will dispatch the message to all registered services.

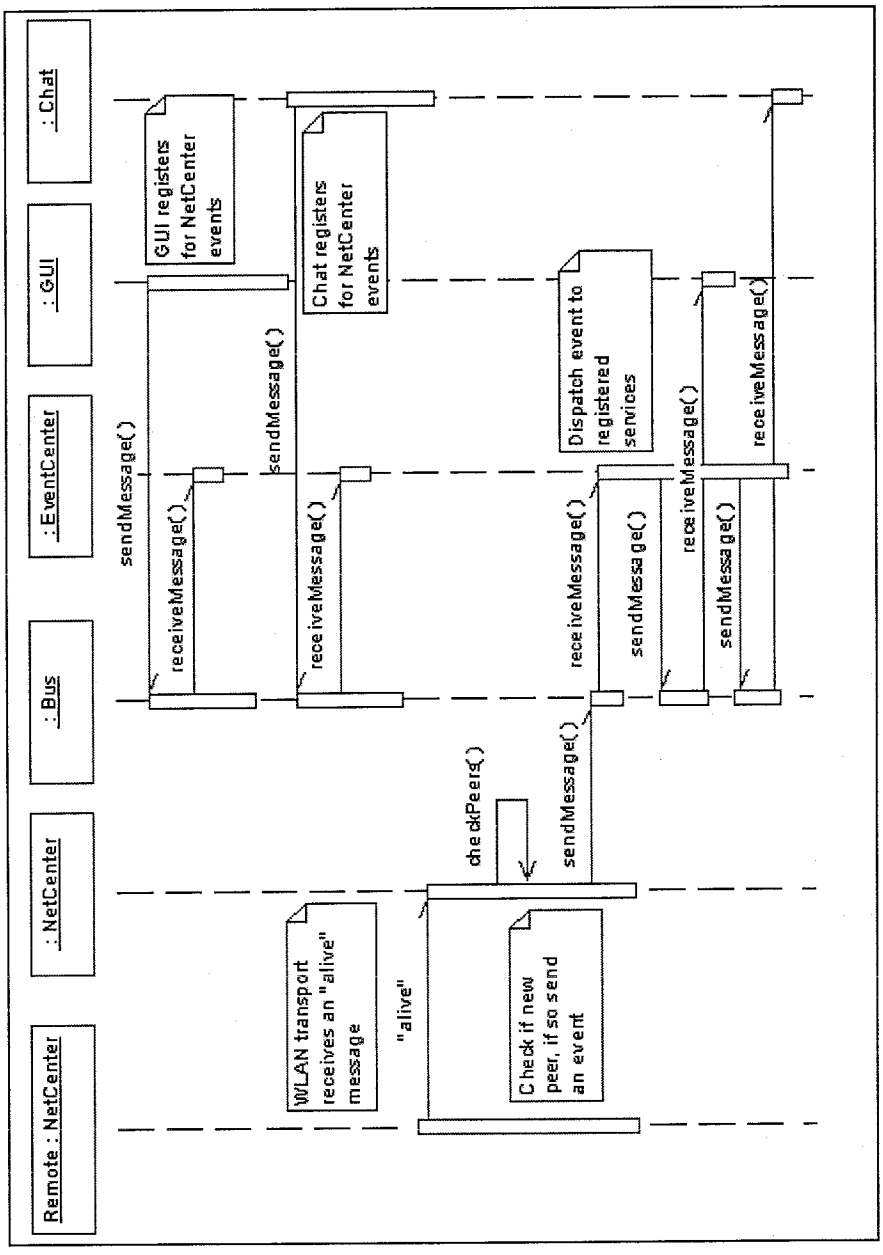

Figure 15: Peer discovered

4.7.3 Peer "offline"

Figure 16 presents the sequence diagram when a peer is timed-out. The peer controller of *NetCenter* periodically checks the list of online peer. If a particular transport has not received an "alive" message (WLAN) from a particular peer, or an "inquire" (IrDA, Bluetooth) does not state this peer is in proximity, it will remove this peer from the list after a number of seconds. *NetCenter* will send an event message to *EventCenter*, which in its turn will dispatch the event to the registered services (Chat and GUI).

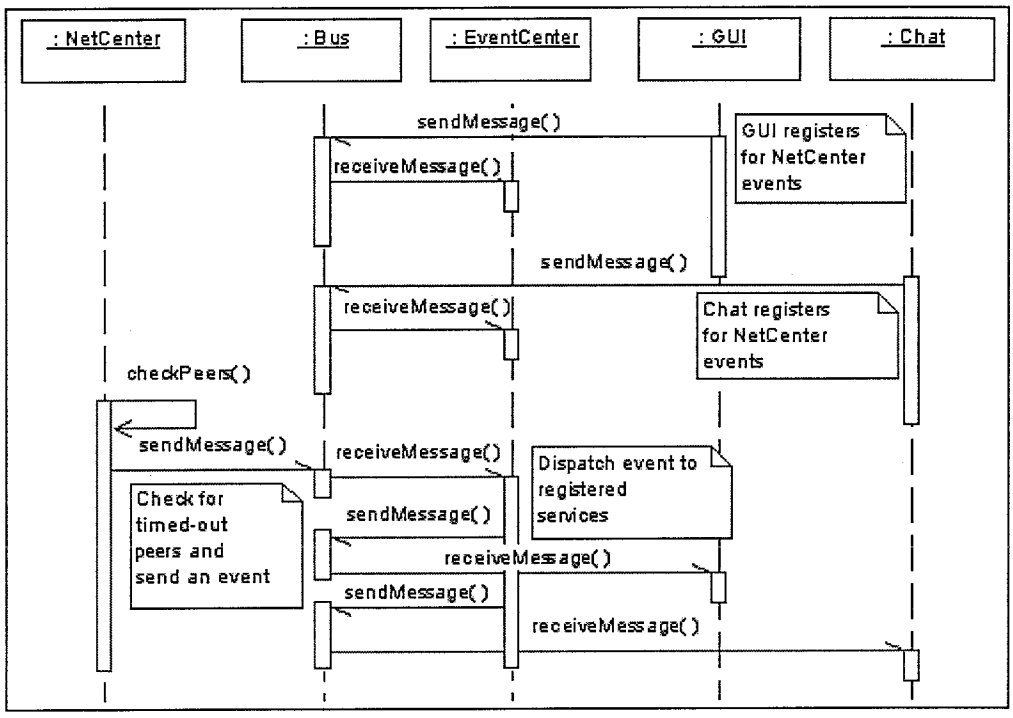

Figure 16: Peer offline

4.7.4 Exploring a Peer

Services can ask peers in proximity which services it provides. Depicted in figure 17 is the sequence diagram for exploring a peer. The *Bus* mediates all communication between local services, but has been left out of the figure for simplicity.

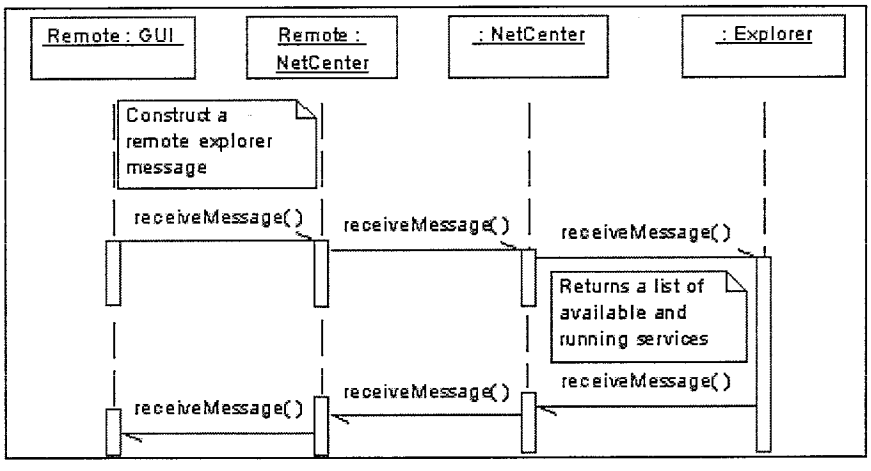

Figure 17: Exploring a peer

A remote *GUI* queries a peer in proximity. *Explorer* will return a list of available and running services. An example of the *Data* field (see figure 9) of an *Explorer Message* is shown in figure 18.

```
<ExplorerMessage>
  <Available>
    <Service>
      <Name>Chat</Name>
      <Description>Talk with your friends in proximity!</Description>
    </Service>
  </Available>
  <Running>
    <Service>Chat</Service>
  </Running>
</ExplorerMessage >
```

Figure 18: Example explorer message

4.7.5 Downloading a Service

After exploring a peer, the user might be interested in downloading a particular service. Below, in figure 19, the sequence diagram for this scenario is depicted.

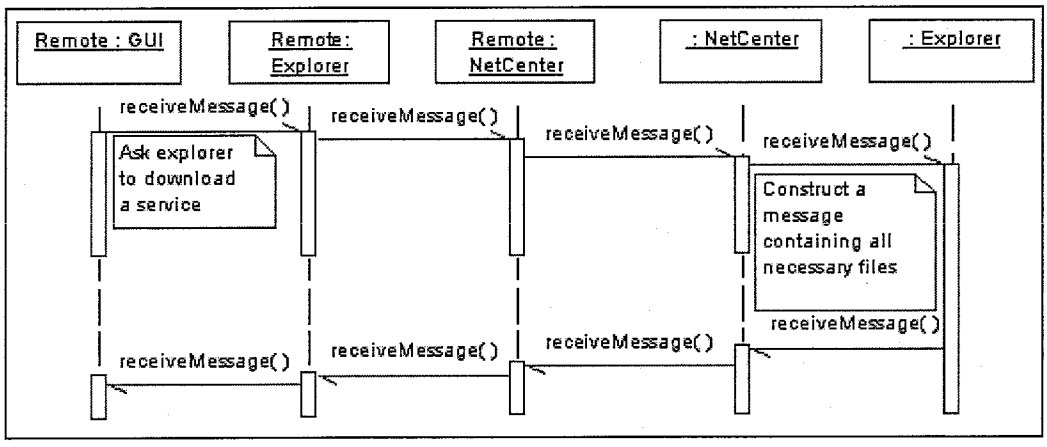

Figure 19: Downloading a service

A remote *GUI* requests *Explorer* to download a particular service. The remote *Explorer* will ask the peer for the service. *Explorer* will construct a *Message* containing all the files (including the config file) necessary to run the particular service and will send it back to the requester. The configuration file might need to be tweaked by the remote *Explorer* before the service can be started (e.g. the execute fields, see figure 10). The remote *Explorer* will store the files (e.g. on disc or in the memory) and notify *GUI* that the new service is available.

4.8 Meeting the Principles and Requirements

This section will provide an overview of how the principles and requirements introduced in section 4.2 are met.

1. *The framework shall be targeted at, but not limited to, PDAs.*
   The core services provide the minimal set of functionalities in order to collaborate. The use of services provides a highly flexible framework that can be blended for different types of PDAs. The use of CORBA causes some extra overhead. However, CORBA provides a mechanism for transparent invocation. The service developer can fully concentrate on programming the logic and does not need to write all the "plumbing" code to establish a connection with the framework. The decision to use CORBA is based on the opinion that these advantages outweigh the disadvantages of the extra overhead.

2. *The framework shall be designed in a way that tolerates the evolution of PDAs and their capabilities.*
   A functionality is captured in a service. Whenever a new functionality is introduced (e.g. a GPS system) it can easily be added to the framework. The event mechanism provided by the framework allows services to take advantage of the added functionality by registering for certain events (e.g. a change in position) coming from this added functionality.

3. *The framework shall use a peer-to-peer communication model to collaborate with its environment.*
   Peers discover each other by "alive" messages or "inquiries" and collaborate directly without using a centralized server.

4. *The framework shall be able to support various types of wireless networks.*
   The different transports provide the required support. Only the core *NetCenter* service will have to be changed when a new type of network is added. The rest of the core services and additional services can stay untouched.

5. *The framework shall be straightforwardly usable by service developers.*
   Although "straightforwardly" is hard to measure, the framework presented is seen as easy to understand. This opinion is based upon a comparison with the frameworks presented in chapter 3. It is out of the scope of this thesis to present this comparison to the reader. Chapter 5 will show how "straightforwardly" services can be developed.

6. *The framework shall enable the discovery of devices and their services in proximity.*
   NetCenter provides the discovery of peers on any type of network supported. After discovery of a peer, the remote *Explorer* can be queried about the services the device offers.

7. *The framework shall enable the use of services, both local and in proximity*
   Local services can use each other with the aid of messages. The use of a remote service is provided by *Explorer*, which can upload and download services.

CHAPTER 5

TRANSPEER

This chapter will introduce TransPeer (transparent peer-to-peer), an implementation of the framework presented in chapter 4. The first section will discuss some implementation issues seen as relevant in the context of this thesis. Section 2 will show how services can be developed.

5.1 Implementation Issues

Platform
TransPeer is fully implemented in Java. It follows the Personal Java 1.2 [29] and Embedded Java 1.1 [25] specifications, which are roughly equivalent to JDK version 1.1.8 [26]. The implementation of the framework has been successfully tested in an environment with several MS Windows 2000 machines, equipped with Sun's J2SE Java Virtual Machine, and Compaq iPAQ PocketPCs (a PDA from Compaq), equipped with Insignia's Jeode Java Virtual Machine [12]. PocketPC is the Operating System developed by Microsoft especially for PDAs.

The framework uses a Java CORBA implementation called HORB [10]. Currently, this is the only CORBA implementation available for PocketPC. Since it is written in Java there was no other option then to implement the framework in Java as well. However, implementations of CORBA will soon be available in other languages (e.g. C++), including HORB.

Future releases of the framework might see a shift from Java to another language (e.g. C++). All the currently available Java Virtual Machines (JVM) for PocketPC use quit a bit of the memory available on the PocketPC. To give an idea, the bare Jeode JVM uses 1 megabyte (without any classes loaded) of the 32 megabytes available on the iPAQ. It must be investigated if other languages can provide a lower memory footprint.

Core Services
All of the core services introduced in chapter 4 are implemented. However, not all services are fully implemented. Table 6 presents an overview of the core services and the functionalities yet to be implemented.

Service	Functionality to be implemented
NetCenter	Bluetooth and IrDA transports
Explorer	Tweaking of the configuration file
EventCenter	-

Table 6: Functionalities to be implemented

Currently, the only transport supported is WLAN. As of this writing the Bluetooth stack has several problems and resulted in no working Bluetooth transport for the framework. However, as soon as a stable version of the Bluetooth stack is available it can be plugged into the Bluetooth transport with ease. The IrDA transport and the tweaking of the configuration file are not implemented due to lack of time. The EventCenter service is fully implemented.

Additional Services
Three additional services have been developed: GUI, Chat and BurgerQueen.

The GUI service provides the user with an overview of the devices in proximity equipped with the framework. Furthermore, it presents the services provided by the devices in proximity. Users can choose to download and start a particular service by using the GUI.

With the Chat service, a user can chat with other users in proximity. Although chatting on a PDA is not optimal due to the lack of a keyboard, it was seen as a service that shows the basic functionality of the framework to the user as well as to the developer.

The BurgerQueen service is developed to show the use of the framework in a commercial setting. It follows many of the ideas presented in the museum guide example mentioned in chapter 1. Potential customers come in proximity of a BurgerQueen restaurant and can browse through the menu. Furthermore, it offers a game that the restaurant can use to convince the potential customer to book a table by giving away free drinks and meals.

Synchronization
As stated in chapter 4, the CORBA implementation used by the framework creates a new thread of the server object (Bus) for each client. As a consequence the Bus process will have *n* threads running when *n* services are connected to it. Access to shared data within the Bus, for example the references to the different services, must be synchronized among the threads. Java provides a solution to this problem with the use of the *synchronized* keyword. Figure 20 presents a code snippet taken from the Bus class.

```
public class Bus {
   //hashtable, service name is key, interface is value
   private Hashtable myServices;

   public synchronized boolean registerService(TransPeerService service, String name ) {
      ...
      //put service in hashtable
      myService.put(name,service);
      ...
   }

   public synchronized boolean sendMessage(Message msg) {
      ...
      //get service
      TransPeerService tps = (TransPeerService) myServices.get(msg.getToService());
      //some checking
      ...
      //send it
      tps.receiveMessage(msg);
      ...
   }
}
```

Figure 20: Code snippet Bus

Shown are the two accessible methods on the Bus for every service. The first method (*registerService*) is called once a service is started to become part of the framework (see figure 14). The other method (*sendMessage*) is called whenever a service wants to communicate with other services (both local and remote). The mechanism that Java uses to support synchronization is the *monitor*. When the synchronized keyword is used on a method, the monitor of the current object is locked. As long as a thread possesses the monitor of an object no other thread can get the lock for that object. The synchronized keyword is therefore used in the Bus class to ensure synchronization of shared data (e.g. the hashtable containing the various registered services).

5.2 An Example Service

This section will show how to build a service for the framework. The Chat service is taken as an example. Not all code lines will be shown, however appendix B provides a full listing of the Chat service.

Since every service is a separate executable unit and is started from the command line by the Explorer service, the first communication between the framework and the newly started service is through the command line. The Explorer passes two parameters on the command line to the service: the path to the IOR file, and the path to the configuration file of the service. ORBs supporting IIOP (such as HORB) identify and publish object references using IORs. An IOR contains the information required for a client to connect to a CORBA object. Every service connects to the Bus using this IOR file.

The interface between the services and the Bus is defined in CORBA IDL. IDL permits interfaces to objects to be defined independent of an objects implementation. After defining an interface in IDL, the interface definition is used as input to an IDL compiler, which produces output that can be compiled and linked with an object implementation and its clients. Every service has to implement the methods defined by the *TransPeerService* interface. This interface defines the methods the Bus can call on every service. The framework only defines one method that has to be implemented, which shows the simplicity of the framework. Appendix B provides a full listing of the IDL used by the framework.

```
public interface TransPeerService {
   public void receiveMessage(Message msg);
}
```

Figure 21: Interface TransPeerService

The listing below (figure 22) presents how this interface is implemented.

```
public class Chat implements TransPeerService {
   private ChatWindow window;

   public void receiveMessage(Message msg) {
      String from = msg.getFromService();
      if (from.equals("NetCenter")) {
         //forwarded the message to the window class, which can easily get all data needed
         window.displayMessage(msg);
      }
      else if (from.equals("EventCenter")) {
         //event message
         //transform into a eventmessage provided by a lib
         EventMessage eventmsg = new EventMessage(msg);
         //get the service that threw the event
         String eventBy = eventmsg.getEventByService();

         if (eventBy.equals("NetCenter")) {
            //netcenter event message, check eventtype
            int eventtype = eventmsg.getEventType();
            if (eventtype == 0) window.peerOnline(eventmsg.getData());
            else if (eventtype == 1) window.peerOffline(eventmsg.getData());
         }
      }
   }
}
```

Figure 22: Implementing interface

Since every service is a separate executable unit, it must provide a *main* method. After initiation, the service will have to register with the framework to become part of the framework. Furthermore, a

method to send a message is provided, used by *ChatWindow* when the user sends a chat string. Figure 23 shows how this is implemented.

```
public class Chat implements TransPeerService {
   private ChatWindow window;
   private Bus_Proxy bus;

   public void receiveMessage(Message msg) {
      ...
   }

   public void sendBusMessage(Message msg) {
      bus.sendMessage(msg);
   }

    public static void main(String args[]) {
        //create Chat
       Chat chat = new Chat(args);
    }

    public Chat(String args[]) {
        //create window
        window = new ChatWindow(this);

        //first argument is the path to the IOR file, which contains the reference to the Bus
        //IOR is a class provided by HORB
        IOR ior = new IOR(args[0]);

        //second argument is the path to the config file
        //parse the config file and set variables such as name
        ...

        //connect
        bus = new Bus_Proxy("iiop://" + ior);

        //register with the bus, name is parsed from the config file
        bus.registerService(name, (TransPeerService) this)

        //register for events from netcenter
        EventMessage event = new EventMessage(EventMessage.SUBSCRIBE, name,
                                                "NetCenter");

        //transform into message that can be sent to bus and send it
         bus.sendMessage(event.getMessage());
    }
}
```

Figure 23: Connecting to the Bus

A CORBA proxy object acts as a proxy for the server object (the Bus). Thus the proxy behaves just like the server object for the client object. Stub routines are included in this object. The object reference of a proxy object is called a *remote object reference*. The server side (Bus) has the counterpart of the proxy object and is called the skeleton object.

The ChatWindow class was not seen as relevant for this example and therefore not shown.

This concludes the example Chat service. The example shows how easy a service can make use of the functionality provided by the framework.

5.3 Summary

This chapter started with some implementation issues. Currently the framework is implemented in Java, however this might change in the future due to memory constraints. TransPeer has been successfully tested in an environment with several MS Windows 2000 machines and Compaq iPAQ PocketPCs. Furthermore, it was shown what is provided as of this writing. The last section showed an example of how easy it is to write a service for the framework.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

This thesis report has described a framework that assists application developers in designing services for PDAs connected in Mobile Ad Hoc Networks. In this final chapter, the research results are summarized. Furthermore, some areas that merit future research are briefly described.

6.1 Conclusions

The research question introduced in chapter 1 reads:

How can we develop a framework that assists application developers in designing services for PDAs connected in Mobile Ad Hoc Networks?

The question was pursued by the identification of requirements to support the building and evolution of collaborative applications. Furthermore, an architecture for a framework intended to provide this support was presented. The implementation of the framework, and an example of how to write collaborative applications using the framework, proved the required support.

6.2 Research Summary

Chapter 1 started with two examples that showed the possibilities of Mobile Ad Hoc Networks. After showing these possibilities, a definition of a framework for collaboration in Mobile Ad Hoc Networks was presented. Also, the benefits of a framework were discussed. Reuse, maintenance and well-known results were mentioned.

Mobile Ad Hoc Networks were discussed in detail in chapter 2. The characteristics and challenges of MANETs were presented. Although the definition of a MANET was introduced just recently, the problems of these highly dynamic networks were already identified long ago. The MANET Working Group is currently working on protocols adjusted to the characteristics of these networks. Additionally, in relation with MANETs, two promising wireless datacom technologies were discussed: Bluetooth and WLAN.

Chapter 3 provided a summary of the related work in the field of frameworks for collaboration. To reason about the different frameworks, the chapter started with the identification of properties to compare. Three well-known architectures coming from the industry were discussed by showing how the frameworks address the various properties. From this study of the related work, some differences between the presented frameworks and our intended framework were mentioned:

- Our framework addresses only one type of device.
- Our framework works in Mobile Ad Hoc Networks.
- Our framework is network independent, whereas two out of three of the mentioned frameworks depend on one type of network.
- Our framework shares services, rather than invokes them.

These differences, due to the envisioned settings in which the framework will be used, were reason enough to design and develop a new framework for collaboration.

The design principles and requirements for the framework were introduces in chapter 4:

- The framework shall be targeted at, but not limited to, PDAs.
- The frameworks shall be designed in a way that tolerates the evolution of PDAs and their capabilities.
- The frameworks shall use a peer-to-peer communication model to collaborate with its environment.
- The framework shall be able to support various types of networks.
- A functionality (or group of functionalities) shall be captured in a service.
- The framework shall be straightforwardly usable by service developers.
- The framework shall enable the discovery of devices and their services in proximity.
- The framework shall enable the use of services, both local and in proximity.

The architecture of the framework was presented thereafter, using the "4+1" view model. The chapter concluded with an overview of how the requirements were met.

In chapter 5, an implementation of the framework called TransPeer was discussed. Furthermore, it was shown how services could be developed with ease.

6.3 Future Research Directions

The field of collaboration in Mobile Ad Hoc Networks has a large scope and this thesis obviously has not addressed all the relevant issues. Presented in this section are some issues that warrant further investigation. Some of them are closely related to issues with software components.

6.3.1 Scalability

The framework presented in this thesis has not been tested in an environment with numerous PDAs. Although the number of PDAs in proximity of a user is limited, there are environments (e.g. class room, conference) where there are many devices in proximity. Research in this area is needed to investigate the impact on the framework when it is used in such environments. Also, research in network protocols is needed since current network protocols do not address the problems of Mobile Ad Hoc Networks (see chapter 2.1). Current standardized protocols for fixed infrastructure networks (e.g. TCP/IP) do not scale well in these mobile environments [13] [14].

6.3.2 Security

The sharing of services poses some serious security issues. Currently, the framework does not provide any security measures for this sharing of services. Although the decision to download services is in hands of the user, the framework could assist the user in making such decisions. As with running an email attachments for example, this decision is currently only based on trust. A security model providing the user with some information about the vendor of the software for example could facilitate the user in making such decisions.

Not only the sharing of services, also Mobile Ad Hoc Networks themselves have security problems that differ from the traditional infrastructure networks. This lack of infrastructure introduces security problems to be solved by researchers [2] [24].

6.3.3 Version Control

As with all software, updates and new versions of the services will be developed and made available for users of the framework. The framework would benefit from a version control mechanism. For example, if a user comes in proximity of someone offering a chat service, the framework could warn the user of differences in version when a chat is initiated. Based upon this information, the user could decide to download the latest version from this user.

6.3.4 Licensing

The sharing of services involves some interesting issues in the field of licensing. To date, applications are bought directly from the vendor or distributor. The framework presented in this thesis introduces a new way of spreading the software. Services can be downloaded directly from users in proximity. Research in this area is needed to investigate if, and how, a licensing model can be implemented in the framework.

Microsoft's Passport of the .NET framework [16] might be a good starting point for this research. Passport provides an authentication mechanism to authenticate users across the Internet using a globally unique identifier. Users can sign in from any device connected to the Internet. Services implementing Passport can authenticate users and could use this information to agree upon a certain license. However, it must be studied if, and how, such a mechanism can be implemented in Mobile Ad Hoc Networks.

REFERENCES

[1] A. Alwan, R. Bagrodia, N. Bambos, M. Gerla, L. Kleinrock, J. Short, and J. Villasenor. "Adaptive mobile multimedia networks." IEEE Personal Communications Magazine, 3(2), April 1996

[2] N. Asokan, P. Ginzboorg, "Key agreement in ad hoc networks", Computer Communications, 2000

[3] Bluetooth SIG, http://www.bluetooth.com

[4] N. Carriero, "Implementation of Tuple Space Machines", Yale University Department of Computer Science, October 1987

[5] S. Corsson, J. Macker, "Mobile Ad Hoc Networking", rfc 2501, January 1999

[6] R. Droms, "Dynamic Host Configuration Protocol", rfc 2131, March 1997

[7] H. Fagrell "Blueportal", Newmad Technologies AB, March 2001

[8] Z. J. Haas. "Panel report on ad hoc networks". Mobile Computing and Communications Review, 2(1), 1997

[9] R. Hinden, S. Deering, "IP version 6 Addressing Architecture", rfc 2373, July 1998

[10] HORB, http://www.horb.org

[11] IEEE Computer Society, "IEEE Recommended Practice for Architectural Description", (IEEE Std 1471-2000), IEEE, New York, 2000

[12] Insignia Solutions, http://www.insignia.com

[13] D. Johnson, "Scalable Support for Transparent Mobile Host Internetworking", Carnegie Mellon University, October 1995

[14] D. Johnson et al., "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols, Carnegie Mellon University, October 1998

[15] P. Kruchten, "Architectural Blueprints – The "4+1" View Model of Software Architecture, IEEE Software 12(6), November 1995

[16] Microsoft Corporation, "Passport .Net", http://www.microsoft.com/partner/products/microsoftnet/passport

[17] Microsoft Corporation, "Plug and Play Specification", http://www.microsoft.com/hwdev/respec/pnpspecs.htm

[18] Microsoft Corporation, "Universal Plug and Play Device Architecture", June 2000

[19] Object Management Group, "CORBA/IIOP Specification 2.4.2", February 2001

[20] B. Pascoe, "Salutation Architecture", Salutation Consortium, June 1999

[21] D. Plummer, "An Ethernet Address Resolution Protocol", rfc 826, November 1982

[22] J. Rekesh, "Coordination frameworks for future networked devices", http://www.cswl.com

[23] J. Rumbaugh, I. Jacobson, G. Booch, "The Unified Modeling Language Reference Manual", Addison Wesley, 1999

[24] F. Stajano, R. Anderson "The resurrecting duckling: Security issues for ad hoc wireless networks", Springler-Verlag, Berlin, April 1999

[25] Sun Microsystems, "EmbeddedJava Application Environment Specification version 1.1", http://java.sun.com/products/embeddedjava

[26] Sun Microsystems, "Java Development Kit version 1.1.8", http://www.sun.com/products/jdk1.1

[27] Sun Microsystems, "JavaSpaces Service Specification", October 2000

[28] Sun Microsystems, "Jini Network Technology", http://www.sun.com/jini

[29] Sun Microsystems, "PersonalJava Application Environment Specification version 1.2", http://java.sun.com/products/personaljava

[30] Sun Microsystems, "Remote Procedure Call Protocol Specification Version 2", rfc 1831, August 1995

[31] Universal Plug and Play Forum, "Universal Plug and Play", http://www.upnp.org

[32] W3C, "Extensible Markup Language (XML) version 1.0", http://www.w3c.org

[33] H. Zimmermann, "OSI Reference Model – The ISO Model of Architecture for Open Systems Interconnection", IEEE Transactions on Communication, April 1980

APPENDIX A

GLOSSARY

A	
Ad hoc network	Self-organizing, rapidly deployable network.
Architecture	The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution [11].
ARP	The Address Resolution Protocol is used to translate an IP address to an ethernet MAC address.
AutoIP	Enhancement to DHCP that allows devices to claim IP addresses in the absence of a DHCP server or similar IP configuration authority.
B	
Bluetooth	Bluetooth is a computing and telecommunications industry specification that describes how devices can easily interconnect with each other using a short-range wireless connection [3].
D	
Datacom networks	Often used to refer to a connectionless network modeled after the postal system. Each message carries the full destination address, and each one is routed through the system independent of all the others (as opposed to telecom networks). Examples of wireless datacom networks include Bluetooth and IEEE 802.11b.
DHCP	The Dynamic Host Configuration Protocol is a mechanism for providing devices with configuration information needed to access the Internet.
Discovery	The act of finding a service.
DNS	Domain Name Service is an Internet service that translates domain names into IP addresses.
E	
IEEE 802.11b	A specifications developed by the Institute of Electrical and Electronics Engineers (IEEE) for Wireless LANs.
F	
Federation (Jini)	A collection of services.
Framework	The foundation on which applications can be built and run.
G	
GPS	Global Position System provides position information. The system gets the positioning information from satellites.
H	
HTTP	The Hypertext Transfer Protocol (rfc 1945). Application-level protocol for distributed, collaborative, hypermedia information systems.
L	
Lookup service	A service that can be used to register to become a member of a federation. Furthermore, it can be used to find services of interest.
M	
MAC Address	Media Access Control Address. Unique physical address of a network interface.
MANET	Mobile Ad Hoc Network [5]. A self-organizing, rapidly deployable network, containing wireless nodes, which may be mobile.

Middleware	Term for a mechanism that serves to "glue together" or mediate between two separate or more programs.
Multicast	An IP Multicast is a mechanism for sending a single message to multiple recipients.
P	
PDA	Personal Digital Assistant, which refers to any small mobile hand-held device that provides computing and information storage and retrieval capabilities.
Peer-to-peer	Communications model in which each party has the same capabilities and either party can initiate a communication session
Piconet	A network of Bluetooth enabled devices.
Proximity	In the range, so that communication is possible.
Proxy (Jini)	An intermediary to contact a service using RMI.
S	
SLM	Salutation Manager. Central component that is used to advertise and lookup services.
SSDP	The Simple Service Discover Protocol is the Universal Plug and Play proposal for how to perform simple discovery.
T	
Telecom networks	Often used to refer to a connection-oriented network. First a connection is established, then data can be passed through (like a tube) and then the connection is released.
Third Generation Wireless (3G)	Third Generation Wireless or 3G refers to near-future developments in personal and business wireless technology, especially mobile communications.
U	
Ubiquitous Computing	The third wave in computing. First wave was the use of mainframes, each shared by high volume of people. Now the personal computing era, person and machine in a one-one relation. Next comes ubiquitous computing, which refers to a "one person"-many-"machines" relation.
UDP	User Datagram Protocol. Connectionless protocol. Unlike TCP, which is a reliable protocol, there is no guarantee that UDP datagrams ever reach their intended destination.
Unicast	An IP Unicast is a mechanism for sending a single message to a single recipient
W	
Wireless LAN	Network to which a mobile device can connect to through a wireless connection

APPENDIX B

LISTINGS

```
import horbx.corba.iiop.*;
import newmad.transpeer.lib.*;
import newmad.transpeer.corba.*;
import java.io.*;

public class Chat implements TransPeerService {
   private ChatWindow window;
   private Bus_Proxy bus;

   public void receiveMessage(Message msg) {
      String from = msg.getFromService();
      if (from.equals("NetCenter")) {
         //network message
         //transform into a netmessage provided by a lib
         NetMessage netmsg = new NetMessage(msg);
         //forwarded the message to the window class, which can easily get all data needed
         window.displayMessage(netmsg);
      }
      else if (from.equals("EventCenter")) {
         //event message
         //transform into a eventmessage provided by a lib
         EventMessage eventmsg = new EventMessage(msg);
         //get the service that threw the event
         String eventBy = eventmsg.getEventByService();

         if (eventBy.equals("NetCenter")) {
            //netcenter event message, check eventtype
            int eventtype = eventmsg.getEventType();
            if (eventtype == 0) window.peerOnline(eventmsg.getData());
            else if (eventtype == 1) window.peerOffline(eventmsg.getData());
         }
      }
   }

   public void sendBusMessage(Message msg) {
      bus.sendMessage(msg);
   }

   public static void main(String args[]) {
      //create Chat
      Chat chat = new Chat(args);
   }

//continued on next page
```

```
    public Chat(String args[]) {
        //create window
        window = new ChatWindow(this);

        try {
            //first argument is the path to the IOR file, which contains the reference to the Bus
            //IOR is a class provided by HORB
            IOR ior = new IOR(args[0]);

            //second argument is the path to the config file
            //parse the config file and set variables such as name
            XMLParser parser = new XMLParser();
            parser.parseFile(args[1]);
            name = parser.getElement("Name");

            //connect
            bus = new Bus_Proxy("iiop://" + ior);

            //register with the bus, name is parsed from the config file
            bus.registerService(name, (TransPeerService) this)
        }
        catch (Exception e) {
            //unable to connect
            System.err(e.toString());
            System.exit(1);
        }

        //register for events from netcenter, will return all current online users
        EventMessage event = new EventMessage(EventMessage.SUBSCRIBE, name,
                                                "NetCenter");

        //transform into message that can be sent to bus and send it
        bus.sendMessage(event.getMessage());
    }
}
```

Figure 24: Listing Chat service

```
typedef sequence < octet > OctetArray;

module newmad {
  module transpeer {
    module corba {
      interface Message{
        void setStringData( in string data);
        void setByteData( in OctetArray data);
        void setCommand( in long cmd);
        void setFromService( in string fromService);
        void setToService( in string toService);

        string getFromService();
        string getToService();
        long getCommand();
        OctetArray getByteData();
        string getStringData();

      };

      interface TransPeerService {
        void receiveMessage( in Message msg );
      };

      interface Router {
        boolean registerService( in string name, in TransPeerService tps);
        boolean sendMessage( in Message msg );
      };
    };
  };
};
```

Figure 25: IDL of TransPeer