NTNU
Innovation and Creativity

Bluetooth enabled Peer2Peer services in ActorFrame

Stephan Søreng Kristiansen

Master of Science in Communication Technology
Submission date: June 2006
Supervisor: Rolv Bræk, ITEM
Co-supervisor: Geir Melby, Ericsson

Norwegian University of Science and Technology
Department of Telematics

Problem Description

ServiceFrame is a framework for developing and execution of services targeting both the internet and mobile domain. The framework is implemented in Java and it runs on J2EE, J2SE and MIDlet (Java enabled mobiles). It supports a PeerToPeer type of services with asynchronous communication between the components running on the different platforms. The communication between the mobile phone and servers or to other mobile phones uses IP over GPRS. Downloading information such as media files is expensive and it takes too long time. A trend is that users of mobile phones want to establish ad hoc network with other users or servers to exchange information.

This thesis is a study of ServiceFrame and how ServiceFrame can be used in a beneficial way to establish ad-hoc networks and service sessions between users of ServiceFrame based mobile devices using Bluetooth. How a ServiceFrame based server acting as a Home Gateway can be used to provide services such as playing music stored on mobile phones shall be studied. The thesis should contain a study of how the different ServiceFrame based devices may exchange information about available services. Existing routing protocol should be studied. Other protocols which could work better in an ad-hoc network may be proposed.

A limitation of the Bluetooth technology is the distance factor between devices, and this study shall look into how ServiceFrame based mobile phones can be used to extend the distance between the source and the target in a communication using mobile devices as mediators. Therefore a study of utilizing ServiceFrame as a base for multihop ad-hoc networking should be done - where a prototype of a service to demonstrate the proposed solutions shall be developed and demonstrated.

Assignment given: 16. January 2006
Supervisor: Rolv Bræk, ITEM

PREFACE

This master's thesis was written at the Norwegian University of Science and Technology (NTNU) as part of the Master of Technology study. It was carried out at the Department of Telematics spring 2006.

I would like to thank my advisor Geir Melby for his involvement, especially with reading through the thesis and giving important advice.

Trondheim, 09.06.2006

Stephan Søreng Kristiansen

ABSTRACT

To make services for mobile devices more user friendly wireless communication is a very helpful tool. Wireless communication normally requires no or very little user input, and communication over a wireless interface is therefore in many cases preferred. A trend today is for example that radio or TV shows are recorded to your mobile device during the day. When arriving your home/office the media file can be transferred or streamed to your desktop computer and/or stereo for better sound and larger screen. Normally this is done using a docking station, but this could also be done through a wireless interface. Combining wireless communication with more advanced service logic opens the possibilities for easier file sharing between mobile devices in a peer-to-peer fashion.

This thesis evaluates ActorFrame as a framework for service creation in Bluetooth ad hoc networks. Through the thesis limitations in the Bluetooth protocol and mobile devices supporting the Bluetooth protocol are discovered. One of the known limitations of Bluetooth is the range limitation, since Bluetooth only is a short range protocol. To extend the range between Bluetooth enabled devices they must be able to function as mediators. For this to be possible information regarding available services on the different devices must be exchanged. This task is performed by routing protocols. The existing routing protocol in ActorFrame is evaluated, and a study of alternative wired and wireless routing protocols is also done.

Besides user friendliness a service should be as cheap as possible. This way as many people as possible will use the service. The Bluetooth functionality integrated in the framework evaluated in this thesis could be utilized in a beneficial way. For example the functionality could be used in peer-to-peer kind of services mentioned above, where a media file is transferred from the mobile device to the desktop computer. Other wireless technologies could also be used for this purpose, namely GPRS or UMTS. As opposed to Bluetooth transmission, which is free, data transmission using either of these two technologies cost money to use. If ActorFrame could be used in a beneficial way depends on the performance of the Bluetooth functionality. Performance below what a certain service demands, where UMTS can offer the required performance quality, would imply

that the framework could not be used in a beneficial way. To test this performance the possibilities of streaming MP3's via the Bluetooth interface using ActorFrame based devices is studied, where a prototype is designed. Through this study limitations both regarding mobile devices running applications based on the ActorFrame framework and the framework itself are discovered.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

AS	**Autonomous System**
IGP	**Interior Gateway Protocol**
BN	**Border Node**
BGP	**Border Gateway Protocol**
EGP	**Exterior Gateway Protocol**
IR	**Internal Router**
DVP	**Distance Vector Protocol**
LSP	**Link State Protocol**
RIP	**Routing Information Protocol**
IGRP	**Interior Gateway Routing Protocol**
LSA	**Link State Advertisement**
OSPF	**Open Shortest Path First**
UDP	**User Datagram Protocol**
JVM	**Java Virtual Machine**
MIDP	**Mobile Information Device Profile**
UML	**Unified Modeling Language**
GSM	**Global System for Mobile Communications**
UMTS	**Universal Mobile Telephone System**
GPRS	**General Packet Radio Service**
IP	**Internet Protocol**
TCP	**Transmission Control Protocol**
MANET	**Mobile Ad-hoc Network**
J2ME	**Java 2 Micro Edition**
J2EE	**Java 2 Enterprise Edition**
J2SE	**Java 2 Standard Edition**
AMS	**Application Management Software**
MMAPI	**Mobile Media API**
JSR	**Java Special Request**
BSF	**Bluetooth Scatternet Formation**
WLAN	**Wireless Local Area Network**

1 Introduction

1.1 Background

The mobile phone technology and mobile phone usage from when it first came in the early 1990's, when the GSM (Global System for Mobile Communications) network was up and running, to this date has had an quite extraordinary development. In less than twenty years, mobile phones have gone from being rare and expensive pieces of equipment used mainly by businesses to a pervasive low-cost personal item. The mobile phones actually outnumber land-line telephones in many countries. Actually it is not uncommon for many people to own a mobile phone instead of a land-line for their residence. For many people a life without the mobile phone is unthinkable. The mobile phone has been a very helpful tool in developing countries where little existing fixed-line infrastructure exists, and has therefore become widespread in these countries.

The number of users and mobile phones sold has had an exploding development, and in that respect development of new services has been very important. Services like SMS (Short Message Service) and MMS (Multimedia Message Service) are services that have caught on in most countries. In addition to different services for mobile phones, the integration of many features on the mobile phone has been important. From being a device only used for communication through speech the integration of digital cameras and capabilities of playing (and displaying) several media files, both sound and video, has turned the mobile phone into a multimedia device. The mobile phone is not only a phone anymore – it is potentially also a digital camera and your portable media player. These capabilities have been possible to realize since the size of the hardware components needed for storage, cameras and playback is reduced "by the minute". With regards to this thesis the importance of making services that transfers media files to and from the mobile device is highlighted. User friendliness and the time it takes to transfer media files are two important aspects when considering services. Focus on how to get media files transferred cheap, fast and easy is of great importance when designing such services, since an average user does not want to use much money on it nor would he/she want to use too much time.

Transferring files from the mobile phone to a desktop computer or transferring files to the mobile phone could easily be done just using a cable. Other more advanced services, however, require other ways of communication. To enable the possibility to send information to and from the mobile phone in a periodic manner or for example when entering your home or office more advanced service logic is required, and using wireless communication is much more practical and increases user friendliness. A trend today is for example to download different radio shows to the mobile device, and listen to the show on the radio on your way to work, on your way to the gym or wherever you are going. The radio/TV show could also have been recorded to your mobile phone during the day and one can come home and transfer the media file containing the show to the desktop computer, enabling automatic playback on a better sound system/bigger screen. This is now commonly done using a docking station connected to the desktop computer. If this could be done automatically without using the docking station when entering your home or office through a wireless interface user friendliness would increase, enabling more people to use such a service. In addition services for peer-to-peer networking could be developed, allowing easier file sharing between several users.

A wireless possibility which commonly is used when downloading information to and from mobile phones is the Internet Protocol (IP) over General Packet Radio Services (GPRS) or Universal Mobile Telephone System (UMTS). This technology can easily send information periodically having some sort of agent realized in a software module installed on the mobile device making a GPRS/UMTS connection periodically to download or upload information. One of the greatest limitations with the technology is economy, since GPRS/UMTS traffic is not free. Another limitation is the limited bandwidth available for GPRS and UMTS, especially GPRS. Transferring media files of a considerable size is then a time consuming process. To make services that respond to location is also possible to realize by using GPRS, but also here the economy and time are parameters that are limitations - making GPRS/UMTS a technology not optimal for the mentioned type of services.

On modern mobile phones one has other wireless communication possibilities such as infrared and Bluetooth. Potentially these technologies can be both faster and cheaper than IP over GPRS or UMTS, but have a significant range limitation that somehow must be overcome. To make services that include networking one can make the services from scratch, or one can utilize frameworks with already implemented modules for networking and behavior. Such a framework exists, a framework called ActorFrame. The framework has modules for networking through the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). As a result of the project assignment done by the author fall 2005 the framework now has integrated Bluetooth functionality as well. In addition it has built in behavior to make new objects, called Actors, which together can form services. These objects have their own addresses that are unique global addresses making it possible to send and receive information between different devices to the specified objects.

By using the newly integrated Bluetooth capabilities, services using wireless communication can be implemented. To overcome the range limitation a self-configuring network of mobile devices acting as routers (and mediators) connected by wireless links, called a multihop mobile ad-hoc network (MANET) can be formed. When comparing wireless to wired communication there are issues of concern regarding for example distribution of routing information. This issue comes quite naturally because devices communicating over a wireless media without infrastructure will produce more topological changes as the devices move in and out of range of each other, compared to communication over wired lines. Communication over wired lines assumes that the communicating devices are more or less stationary, compared to the mobile view of a wireless communication protocol.

The thesis will look into the issue of routing protocols used in communication over wired lines versus wireless environments, and will also look into the routing protocol used in the ActorFrame framework for distributing routing information. Since the objects that perform services are addressable a routing protocol actually is as a way of exchanging information about available services in the network. The question of whether or not the

ActorFrame framework is suitable for ad hoc networking using the newly integrated Bluetooth functionality is also an important part of this thesis. Generally speaking this thesis is an evaluation of ActorFrame performance in a wireless ad-hoc environment.

1.2 The thesis

In this part the goals of the thesis will be outlined. In addition the resources used during the work on the thesis are presented.

1.2.1 The goals

This assignment is given by Ericsson AS and it mainly proposes a study of ServiceFrame and how ServiceFrame can be used "in a beneficial way to establish ad-hoc networks and service sessions between users of ServiceFrame based mobile devices using Bluetooth". To easily show the main goals of this thesis a list of goals can be given:

1. Study of the routing protocol used to distribute routing information between ServiceFrame based devices. In addition a study of alternative ways of exchanging information on available services should be done.
2. A prototype of a service to demonstrate proposed solutions shall be developed and demonstrated. Since a study on how playback of a song stored on a mobile phone could be possible on a ServiceFrame based device acting as a Home Gateway should be done, the prototype will demonstrate this. Included in this study possible ways to realize the playback functionality is implied as an additional study.
3. Decide whether or not ServiceFrame can be used "in a beneficial way to establish ad-hoc networks and service sessions between users of ServiceFrame based mobile devices using Bluetooth".
4. A study of whether or not ServiceFrame can be used to extend the range limitation that Bluetooth implies using ServiceFrame based devices as mediators.

The original assignment description giving a more thorough outline of the given goals is given on the front page of the thesis.

1.2.2 Resources

The assignment will be carried out at the PATS lab (Program for Advanced Telecom Service) [1] located at the department of Telematics at NTNU, which was opened November 2001. It is a teleservice laboratory that aims to "provide a setting in which students and researchers can experiment with the development of advanced telecommunication services"[1]. The assignment will use the ActorFrame and MIDletActorFrame framework [2], which together with the Java Bluetooth API (Application Programming Interface) [3] will provide communication both using Bluetooth and more "normal" wired communication using TCP and UDP.

To do a study of a multihop ad-hoc network possible, multiple mobile devices were required. Therefore two Sony Ericsson P900 [4] were used:

Figure 1: Mobile phone used to make multihop ad-hoc network[4]

The P900 supports a lot of features described in [4], but especially important is support for Bluetooth, Java 2 Micro Edition (J2ME) and Java Special Request 82 (JSR82) – since this is required to run MIDletActorFrame on the device. The Bluetooth technology will be further described in chapter 3, along with J2ME and JSR82.

To Bluetooth-enable the desktop computer a Universal Serial Bus (USB) Bluetooth dongle was used. This was a Belkin class 2 device, which means that the Bluetooth dongle gives the desktop a Bluetooth-range of approximately 10 meters.

In addition access to the newest updates of both ActorFrame and MIDletActorFrame was acquired through Ericsson.

1.3 Assumptions and constraints

One obvious assumption is that the mobile phone that runs the MIDletActorFrame framework needs to have support for the Java Bluetooth API, also known as JSR82. As it turns out, this is not the case for all mobile devices which support Java and Bluetooth.

The P900 only have support for Bluetooth v1.1 and this is a huge constraint regarding bandwidth. This should be kept in mind when examining speed issues later on in the thesis. Testing the Bluetooth functionality with Bluetooth v2.0 compatible devices should have been done, but such devices were not available.

1.4 An outline of the thesis

Chapter 1 gives a short introduction and presents the problem background and assignment description.

Chapter 2 describes Ericsson's service creation environment: ServiceFrame, ActorFrame, JavaFrame and MIDletActorFrame. Special attention is given to ActorFrames' and MIDletActorFrames' router implementation called ActorRouter in ActorFrame and MIDletRouter in MIDletActorFrame.

Chapter 3 introduces the most important technologies concerning the thesis. A brief introduction to the Bluetooth network topology, the Bluetooth protocol stack and Bluetooth profiles is given. Emphasis here will be on Bluetooth functionality in scatternets – what is commonly supported by mobile devices and what should be implemented to realize ad hoc capabilities. In addition Java for small devices, J2ME, is described along with a short outline of Bluetooth for Java and a new optional package for J2ME based devices which will specialize on ad hoc networking.

Chapter 4 describes the issue of routing. Routing protocols both for "normal" wired communication with infrastructure and wireless communication without infrastructure,

also known as mobile ad-hoc networking, will presented here. In this chapter the implemented routing protocol in ActorFrame and MIDletActorFrame will be discussed, along with usability in Bluetooth networking.

Chapter 5 presents the design and implementation of the prototype made to test Bluetooth functionality. Also an outline of different possibilities regarding prototype implementation is given.

Chapter 6 deals with testing of various aspects of ActorFrame.

Chapter 7 discusses the knowledge acquired through the work on this thesis. There is also a brief note on possible future work.

To show how to get the prototype running some notes about this is given in Appendix A, along with Appendix H containing the code to actually be able to run the prototype made in this thesis. In addition other Appendixes containing information important for the thesis, but not appropriate to include in the main text are included.

2 The service creation environment

Telecommunication is by nature asynchronous and for this reason the frameworks JavaFrame [2, 5, 6] and ActorFrame [2, 7] have been developed by Ericsson. In these frameworks state machines and asynchronous communication with messages is important. On top of these frameworks one finds ServiceFrame [2, 8, 9]. This framework uses the mechanisms offered by ActorFrame and JavaFrame to provide "support for service creation, service deployment and service execution" [8]. In addition to these frameworks the MIDletActorFrame framework which corresponds to ActorFrame, designed for small mobile devices, has been developed. MIDletActorFrame has only quite recently been developed and, because of this, documentation for MIDletActorFrame is lacking. In this thesis the routing protocol implemented for both ActorFrame and MIDletActorFrame is in focus and therefore mechanisms and classes handling routing will receive special attention. In that respect some of the following will serve as documentation of undocumented parts of both ActorFrame and MIDletActorFrame.

In this thesis ActorFrame and MIDletActorFrame is most relevant and therefore they will be described first, leaving a smaller description of JavaFrame for the end of this chapter. ServiceFrame is also relevant for this thesis, but as will be explained in chapter 2.4 ServiceFrame is an application of ActorFrame. To fully understand the concepts of ActorFrame and ServiceFrame it is recommended to study [2, 7, 8, 9] as this chapter mostly describes how the network communication modules are built.

2.1 ActorFrame

Since ActorFrame first came in 2002 [2] it has come several new versions with updates coming quite frequently. This thesis will focus on the ActorRouter which deals with routing information in the framework, and also the modules that handles network communication.

ActorFrame is implemented in three different Java packages [7], meant to be used on three different platforms. For the Java Virtual Machine (JVM) platform the regular ActorFrame package exists. EJBActorFrame is a package that runs on a J2EE application

server, and the MIDletActorFrame package runs on small handheld devices that run Mobile Information Device Profile 2.0 (MIDP) on the J2ME platform.

In 2.1.1 general concepts for ActorFrame is given and in the subchapter following 2.1.1 the network communication architecture in ActorFrame is described. Although the concepts and the routing in MIDletActorFrame are very much the same, some points are worth mentioning, given in chapter 2.2.

2.1.1 General concepts

The core concept of ActorFrame is the Actor. Actors are objects with a state machine and an optional inner structure of actors. This is illustrated in figure 2:

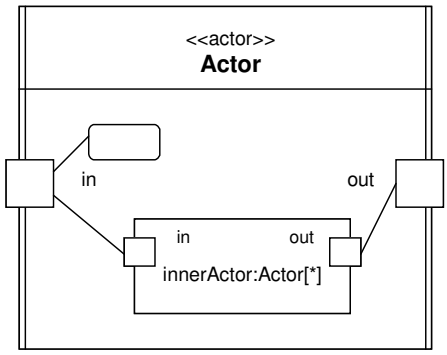


Figure 2: Illustration of the Class Actor with optional inner structure of actors [2]

These inner actors can be static, with the same lifetime as the enclosing actor, and other actors can be dynamically created and deleted during the lifetime of the enclosing actor. The concept that “actors play roles” is supported by ActorFrame. If the actor is to play several roles this is done by creating several inner actors each playing a role. Communication between actors is done by passing messages through the in and out ports, whereas all messages are routed between actors using the ActorRouter.

Actors support the ActorFrame protocol, including protocols such as role request and role release. The main idea is that an actor can request another actor to initiate new roles to do a requested service.

ActorFrame can be described by a Unified Modeling Language (UML) 2.0 profile. The figure below shows the profile package with the stereotypes needed to model ActorFrame:

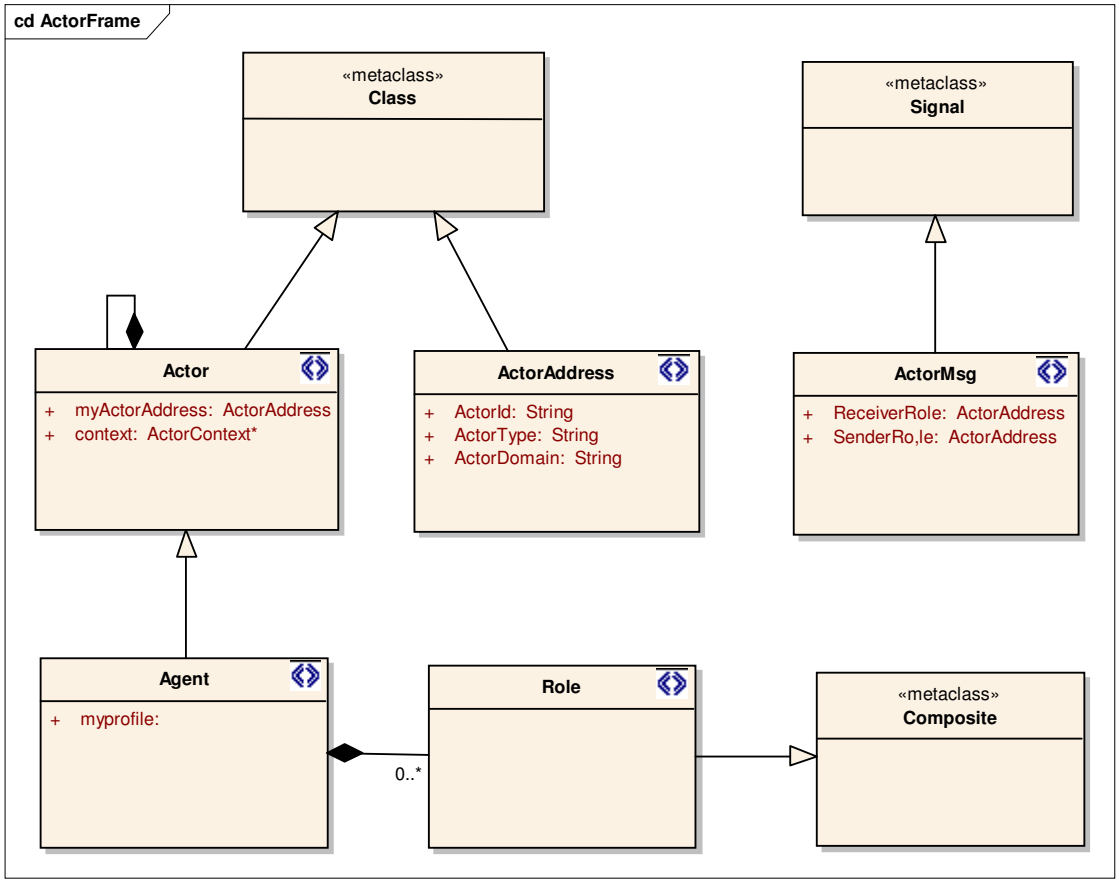


Figure 3: The ActorFrame profile package

The concept of the actor is represented as the stereotype actor. All messages that an actor may send and receive have to be a subtype of the superclass ActorMsg. This message has a sender and a receiver attached, represented by ActorAddresses, which is a unique identifier of an instance of an Actor.

2.1.2 The communication architecture

A complete overview of ActorFrame is not the intent with this subchapter. It is a brief description of the parts of the system concerning the network communication and routing functionality in ActorFrame.

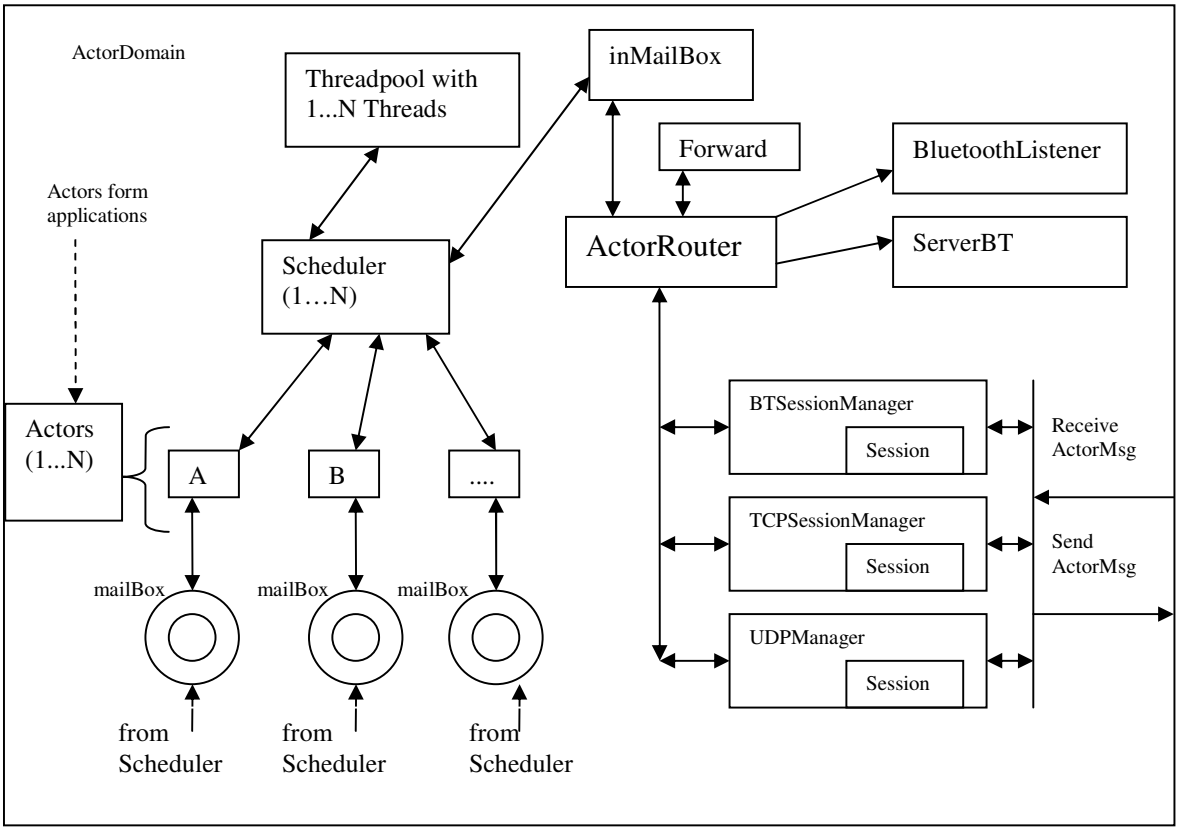


Figure 4: Components involved with the routing in ActorFrame

2.1.2.1 Different parts of the architecture

From the figure above we see the Scheduler, which is connected to one or many Threads from a Threadpool. Each Thread is allocated to an Actor (with an associated state machine) and the Scheduler unit schedules the different actors (and state machines) in the framework - deciding which ones to be active at all times. Information between Actors in one ActorDomain (locally) is passed on through messages, called ActorMsg's, and put in the input queue of the receiving Actor. The Scheduler decides which input queue to put the message in, since the Scheduler can find the references to the correct input queue belonging to the receiving Actor. The input queue is called a mailbox, and every Actor has a mailbox. When messages are put in the mailbox the Actor reads the messages in a first in first out manner.

When the receiving Actor is located on a different ActorDomain (which in most cases also imply that it has a different physical location) messages must be routed and sent

using the networking modules. The knowledge of where other Actors are located is received through the routing protocol implemented in ActorFrame, which realizes exchange of routing information between ActorRouters. To give an accurate description of how a message is handled in the Scheduler when messages are sent between ActorDomains, an elaborate outline of the Scheduler module is required – this is not the intent here. It is more appropriate to describe the handling of messages from the moment the message is added, either internally from actors on the same machine or externally from actors on other machines/devices, to the inMailBox (the input queue) of the ActorRouter.

2.1.2.2 The SessionManager modules

Before an outline of how messages are handled in ActorRouter will be given, the internal structure of the SessionManager modules shown in figure 4 should be explained. The ActorRouter contains SessionManagers for TCP, UDP and Bluetooth connections, where the main objective of the SessionManagers is to create sessions. The SessionManager classes are explained below.

TCPSessionManager includes the following inner classes: Session, SessionServer, Reader and Writer. The SessionServer is the module accepting new TCP-connections, running a normal accept() method that kicks in when a request for a new connection is received. Session contains a Mailbox, Reader and Writer, and this class represents a Session that will be open/running as long as the TCP connection is needed. The Reader class receives messages sent from the externally located actors reachable through this TCP connection. The opposite functionality is found in the Writer class, which takes care of transmitting messages. Messages transmitted by a session are put in the Mailbox by the ActorRouter.

BTSessionManager basically includes the same as TCPSessionManager, namely the inner classes SessionBT, ReceiverBT and SenderBT. The server functionality is separated from the inner structure of the BTSessionManager. BluetoothListener and ServerBT are the classes responsible for the Bluetooth functionality. The BluetoothListener listens for new

Bluetooth devices in addition to connect to them. When the BluetoothListener connects it sends a connect message that is received by the ServerBT on the receiving device, which creates a new session. These sessions will, as TCP sessions, be running as long as the Bluetooth device is in range.

The UDPManager have similar functionality as the two manager modules mentioned above.

2.1.2.3 How messages are handled

Having dealt with the various parts of the architecture a rough description of how messages are handled in the ActorRouter is appropriate.

1. When an actor calls the sendMessage() method the Scheduler adds the message to the input queue (inMailBox) of the ActorRouter if the receiver is not located on the same device. Messages are also put in this input queue if the message is received from actors on other ActorDomains, but then the message is added to the queue via the reader module of a UDP, TCP or Bluetooth session object.
2. If the actor to receive the message is known (stored in the forward table belonging to the ActorRouter, shown in figure 4), the ActorRouter checks if the receiver IP-address is the local IP-address; in case it is, the message is put in the message queue belonging to the state machine of the receiving actor. This task is done by calling the output-method of the Scheduler, since it is the Scheduler that has the complete overview of existing state machines in this ActorDomain. This would only happen if the message is received by the TCP/UDP/BT receiving units. Messages internally in the ActorFrame installation are “routed” by the Scheduler, without including the ActorRouter.

   If the IP-address of the receiver differs from the local IP-address the message is forwarded to the actor using the transmitting units located within the session objects controlled by the SessionManagers. If the receiving actor exists in the forward table of the ActorRouter the correct SessionManager and name of the session is extracted from information in the forward element of the forward table.

This enables the SessionManager to find the correct session and thereby letting this sessions' transmitting module send the message to the receiving actor.

If no forward element exists in the forward table for the given receiver the message is sent to the default gateway.

Note that the description of message handling is coarse. Despite the description being coarse important aspects are discovered. Among these discoveries are usage of default routes, mechanisms to prevent messages from floating around in the network forever and there exists different ActorDomains.

The Forward table is built exchanging information between ActorRouters (MidletRouters for devices running MIDletActorFrame). A more elaborate outline of the routing protocol used in ActorFrame will be given in chapter 4.

2.2 MIDletActorFrame

The MIDletActorFrame package makes it possible to use the concepts of ActorFrame on a mobile device. The outline given in 2.1 in connection with figure 4 also applies for MIDletActorFrame. Although there are small differences, the router architecture is now more or less the same for both ActorFrame and MIDletActorFrame. This was not however the case before the Bluetooth functionality was integrated. To clarify this point an outline of this development follows.

2.2.1 Original router architecture for mobile devices

Before the integration of Bluetooth was done the routing architecture in the MIDletActorFrame package was very simple. The router only sent periodical updates of actors located on the mobile device using UDP. Therefore the mobile device did not include any routing table, so that using the mobile device as a mediator was not possible. As figure 5 shows a message then had to be sent to an ActorFrame based machine (typically a desktop computer) which knows the location of the receiver, or a default gateway which hopefully knows the location. If one already have a mobile device in between which has a Bluetooth connection to both Sender and Receiver, this device could

work as a mediator for the message. This way the range limitation of Bluetooth can be overcome.

2.2.2 The enhanced router architecture for mobile devices

Bluetooth functionality introduced the idea of using ServiceFrame based mobile devices in ad hoc networks. For this to be possible the mobile device would have to have common router functionality - like having a routing table and gathering routing information, enabling it to function as a mediator. The figure below shows how a message could be sent through the device acting as the mediator, in a way overcoming the range limitation of Bluetooth. This configuration demands that the device can send and receive routing information and also keep a routing table so that forwarding can be done. The device must also support multiple simultaneous Bluetooth connections, a functionality not supported by all mobile devices (see chapter 6.1)

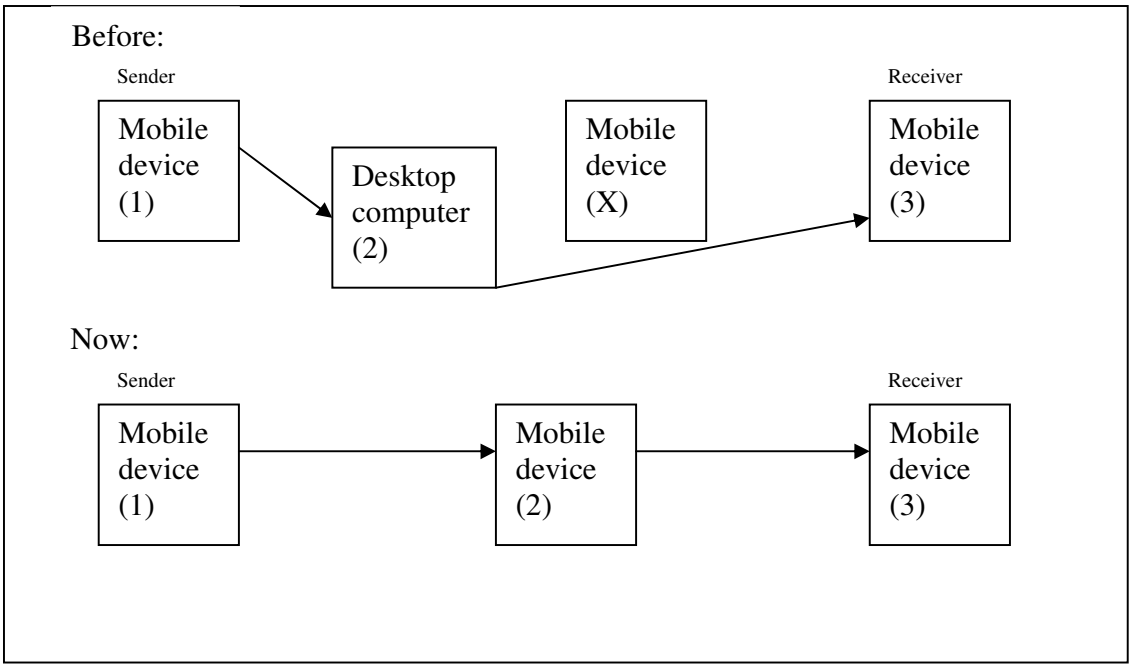


Figure 5: Explaining differences in routing messages before and after Bluetooth integration

2.3 ServiceFrame

This thesis focuses on how ServiceFrame can be used in a "beneficial way to establish ad-hoc networks and service sessions between users of ServiceFrame based mobile devices using Bluetooth". Therefore a brief presentation of ServiceFrame is given.

According to [8] ServiceFrame provides architectural support for service creation, service deployment and service execution, but with no provision of any end user services. Services are realized by ServiceFrame applications that are defined by specializing and instantiating framework classes.

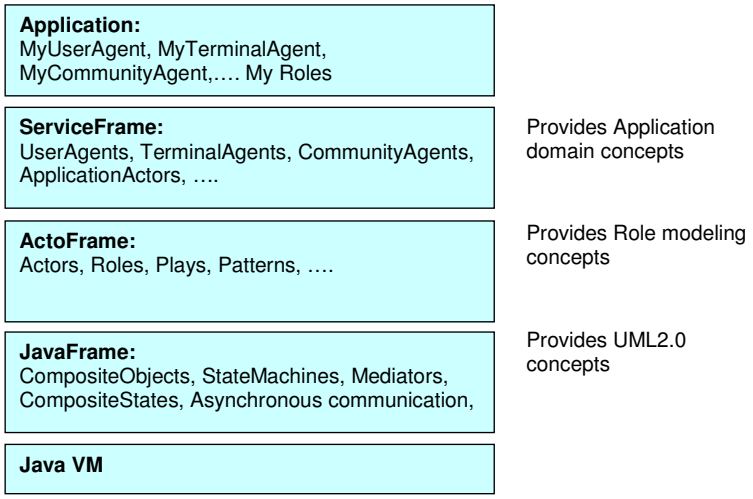


Figure 6: Illustration of the ServiceFrame layers [8]

As can be seen from figure 6 ServiceFrame is layered on top of ActorFrame and JavaFrame, meaning that ServiceFrame actually is an application of ActorFrame. Because of this it is not a fine line between focusing on ServiceFrame or ActorFrame.

ServiceFrame is an application server in the service network [10] and it provides functionality for communication with users connected through different types of terminals such as phones, PC's or PDA's. Access to network resources through the OSA API (Open Services Architecture Application Programming Interface), which enable services to set up phone calls between users, is also provided.

What ServiceFrame provides is the domain given actors in figure 7 shown below, with generic attributes and behavior. These are concrete components that give access to the telecommunication network, through GSM, GPRS, IP-zones and other accesses.

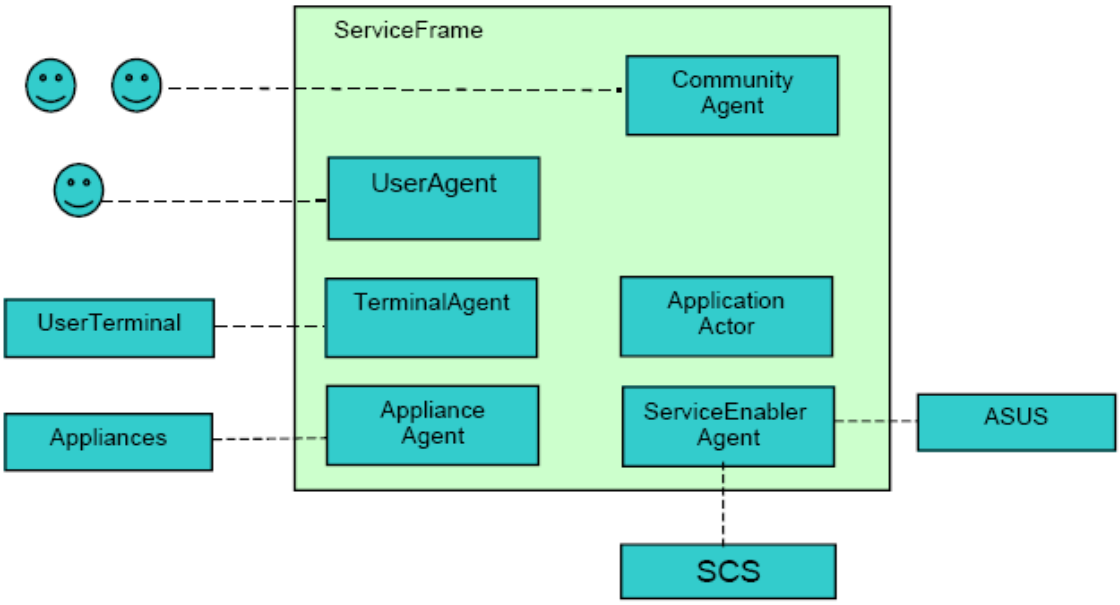


Figure 7: Domain given Actors that mirror the environment provided by ServiceFrame [8]

2.4 JavaFrame

As the title of [5] implies, JavaFrame is a framework for Java enabled modeling. More precise it is "a Modeling Development Kit (MDK) for development and execution of state machines in Java" [2]. The system model should be possible to model in a language independent way, so that it can be analyzed and to get the system model right. In addition implementation should be done in Java. With JavaFrame it is possible to apply modeling techniques and still work in Java, having a one-to-one relationship between the Java source and the model.

In addition to being an execution environment JavaFrame is a library of classes used to implement state machines and asynchronous communication between state machines [2]. The behavior of active objects in the system is described by either state machines or by a structure of interacting state machines.

ActorFrame is an extension of JavaFrame that uses state machines and asynchronous communication provided by JavaFrame. Ports, parts, connectors and the ActorFrame

protocol are some of the additions made in the ActorFrame implementation. This is illustrated in chapter 2.3, figure 6.

3 Bluetooth and Java

There are mainly two technologies involved in this thesis – Bluetooth and J2ME. This chapter will present the technologies and will also unveil some of the constraints concerning both technologies. Some of the text in this chapter is also given in [11] and is included in this thesis for the sense of completeness. This goes especially for chapter 3.1.1, 3.1.6 and 3.1.7. Also chapter 3.2.1 through 3.2.3 is mostly from [11].

3.1 Bluetooth

Bluetooth is a radio communication protocol that was originally envisioned in 1994 by Ericsson as a short range communication protocol for mobile devices [13]. Ericsson teamed up with IBM, Intel, Nokia, and Toshiba and formed a Special Interest Group (SIG) to "develop a royalty-free, open specification for short-range wireless connectivity". The protocol is named after King Harald Blåtand of Denmark, because of his accomplishment of uniting Denmark and Norway.

Bluetooth can act as a replacement for cable, infrared and other connection media, but it also offers a large selection of other services. For example it is a good technology for synchronizing devices. Since routing in mobile ad-hoc networks is an important part of this thesis there will be some emphasis on different network topologies in which Bluetooth takes part. This chapter gives an overview of the Bluetooth protocol and extensions necessary to utilize Bluetooth in multihop networking. The parts taken from [11] are based mostly on chapter 1, 2 and 3 in [13], and also on [12] and [14].

3.1.1 Comparison with other wireless protocols

It is not obvious why one chooses to focus on Bluetooth as the wireless protocol. Therefore, to get a of perspective on the advantages of Bluetooth in small devices, such as mobile phones, this subchapter compares Bluetooth to two of the most known wireless communication protocols besides Bluetooth.

3.1.1.1 Bluetooth versus infrared

Devices communicating over an infrared link must be only a few feet apart, in line-of-sight of each other. The range-limitation is overcome by Bluetooth by having a range of 10 meters for the most common devices. Bluetooth works like a radio and is omni directional, and therefore any line-of-sight issues disappear. A range of 100 meters is achieved by Bluetooth technology, but this requires a much greater power source than the battery of small mobile devices.

3.1.1.2 Bluetooth versus 802.11 standards

802.11 is a collection of standard protocols for wireless LANs (Local Area Network). The design goals for Bluetooth and 802.11 are different although they both operate in the same frequency band (2.4GHz). The 802.11 standard connects larger devices (laptops, desktops) that have a more powerful source at high speeds (11 Mb/s for 802.11b) over larger distances, while Bluetooth is intended to connect smaller devices like mobile phones and PDAs (Personal Digital Assistant) with connection speed up to 2-3 Mb/s and distances of up to 10 meters. This allows Bluetooth to be a low power technology, which make a huge difference in the battery life time of mobile devices. Despite these advantages Bluetooth cannot replace 802.11 in large file transfers and long-range communication.

3.1.2 Specifications

Bluetooth operates, as mentioned, in the same frequency band as the 802.11 standards. The radio frequency is centered at 2.45 GHz, with 79 channels spaced with 1 MHz. This means that the radio frequency (RF) channels are 2420+k MHz, where k = 0 to 78 [15]. Devices that could cause interference with Bluetooth operation are devices operating in the same radio band, which could be other Bluetooth devices, devices following the IEEE 802.11 standard, Microwave ovens, cordless phones, licensed users and microwave lighting [16]. To reduce interference Bluetooth utilizes a frequency hopping scheme. Bluetooth uses adaptive frequency-hopping spread spectrum (AFH) to avoid using crowded frequencies in the hopping sequence.

Bluetooth can support three full-duplex voice channels simultaneously in each piconet, which is a collection of Bluetooth devices that is synchronized to the same hopping sequence [17]. Data rates using Bluetooth depends on which version of Bluetooth that is used. Using v1.2 a maximum data rate of 721 kbps (kilobits per second) is supported. If version 2.0 is supported data rates up to 3 times faster is supported giving a data rate of 2-3 Mbps (Megabit per second). [18].

3.1.3 Architectures

There are two important terms regarding Bluetooth network architecture, namely piconets and scatternets. An important aspect of ad hoc networking with Bluetooth devices is connectivity. The Bluetooth standard allows creation of collections of piconets, which is called scatternets. Creation of scatternets is not, however, specified with a specific protocol in the Bluetooth specification. This issue will be dealt with in the subchapter concerning scatternets.

3.1.3.1 Bluetooth piconet

As mentioned in 3.1.2 a Bluetooth piconet is a collection of Bluetooth devices synchronized to the same hopping frequency. This collection could also be called a cluster or a grouping of Bluetooth devices. By having this clustering, strategy coordination at the media access layer and routing is allowed.

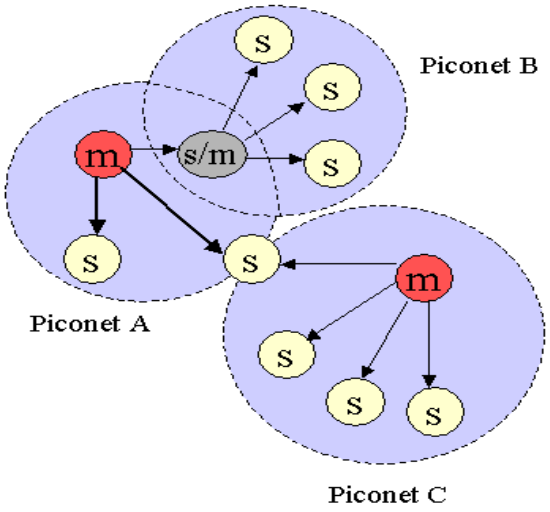


Figure 8: Illustration of Bluetooth network topology. An important point is that a master in one piconet can be a slave in another. Another point is that a device can be master in at most one piconet. [12]

The illustration from figure 8 shows that each piconet has a cluster head, which is called the master (m), which acts as the central controller. All other devices within the piconet become slaves (s). In a piconet there can be no more than 8 active devices, thus there will be at most 7 slaves in addition to the master. If there are other devices associated by the piconet in addition to these 8 devices, they can not be active – hence they will be considered "parked".

The piconet adheres to the same hopping frequency and timing as the master. Slaves of the piconet only have links to the master, meaning that all communication in a piconet goes through the master.

A slave can act as a master in another piconet, whereas a master only can act as a master in one piconet at a time. If a Bluetooth device is within the locality of two piconets the formation of a scatternet can occur.

3.1.3.2 Bluetooth scatternet

Although the Bluetooth standard allows creation of a collection of connected piconets, also known as scatternets, it does not give any particular protocol for doing so [19]. Scatternet support is also optional for vendors to implement. Mobile phone vendors have to this date chosen to not implement scatternet support in their chipsets, but for vendors of other Bluetooth products, such as USB dongles, scatternet support is more widespread. In 3.1.5 it is outlined what vendors must implement to realize scatternet support. In 3.1.4 protocols for creating (and maintaining) scatternet formations are presented, and hence the subchapter describes one of the things that must be implemented to create and maintain a multihop ad hoc network of Bluetooth enabled devices.

3.1.4 Bluetooth Scatternet Formation

To be able to create a connected multihop ad hoc network of Bluetooth devices a Bluetooth scatternet formation (BSF) protocol must be implemented. Without a well implemented BSF protocol traffic throughput will be low and network topology change will be poorly handled. In ActorFrame the only "protocol" handling this task is that a device running the framework and which is Bluetooth enabled will try to connect to every

other ActorFrame based device in range. This is not a very sophisticated protocol, and it does not scale well. Therefore a more sophisticated protocol should be implemented, like one of the protocols discussed in this subchapter. No implementation of a BSF protocol has been done on mobile devices because scatternet operation is yet to be included in the chipsets.

Among the main decisions necessary for such a protocol is the question of which role a node should play, which is whether a node should be a master, a slave or both. Since an important aspect of this thesis is to study how ServiceFrame can be utilized as a base for multihop ad networking, and as mentioned above there does not exist any sophisticated protocol for forming and maintaining the Bluetooth network topology created by the framework, BSF protocols should be studied and described.

There exist many BSF algorithms whose main goal is to provide a connected and degree limited scatternet topology [19]. Degree limited means that the protocol always makes topologies where the master never has more than seven slaves. A protocol can also be centralized, where one node distributes master-slave decisions. Another possibility is to use a distributed algorithm, which can be further classified into globalized and localized protocols. In localized protocols each node make formation decisions based only on the information from its neighbors, whereas the globalized ones use information for the complete topology. A scatternet formation protocol should also have a self-healing process involved, so that information about nodes leaving and entering the network is updated whenever a change is detected.

The design criteria for the formation protocols vary, but it is anticipated that scatternet formation protocols should have at least the following goals in mind [19]:

- minimization of the number of piconets, and therefore also minimization of the number of master nodes
- minimize the number of slave roles each node has (be a slave in a minimum number of piconets inside the scatternet formation)
- minimize the number of master-slave nodes (nodes that are both master and slave)

There exists various metrics that can be used for scatternet evaluation, depending on the main performance goal of this scatternet. It could depend on maximum capacity, path lengths, node availability and many other factors, and would change depending on what the scatternet would be used for.

[19] present possible solutions available as scatternet formation protocols. There are different solutions depending on whether scatternet formation is to take place in single-hop networks or multihop networks. Some of the solutions will be presented to more detail than others.

3.1.4.1 Bluetooth scatternet formation in single-hop networks

In a single-hop network all the wireless devices are in the radio vicinity of each other. When the scatternet is formed the single-hop network is converted into a multihop scatternet. An article given by Stojmenovic and Zaguia ([19]) presents many possible BSF protocols. Below several types of protocols are mentioned, mostly to show how many different kinds of protocols one potentially can choose from.

Centralized BSF protocols

- *Traffic and capacity based scatternets*
- *Super-master election for central decisions*
- *Ring topology*

Distributed BSF protocols

- *Tree scatternet structure*
- *Mimicking known topologies*
- *Minimal spanning tree based scatternets*
- *Loop scatternet structure*
- *On-demand scatternet formation and maintenance*

The *ring topology* [20] for scatternet formation in single-hop networks is very simple and can easily create scatternets. A ring is created by the master nodes and slave-slave bridges (like the slave in on the borderline between piconet A and piconet C in figure 8). Because the protocol is centralized it is not scalable, which limits the number of nodes that can be

involved in the scatternet formation. It is beyond the scope of this thesis to go into detail of all the different types of protocols, hence BSF protocols for multihop networks (which are more interesting for this thesis) are presented next.

3.1.4.2 Bluetooth scatternet formation in multihop networks

In multihop networks some of the wireless devices are not in range of each other. As for BSF in single-hop networks there are some *centralized protocols* with one central node making all the decisions. Centralized scatternet formation does, as mentioned, not scale well and therefore little focus on these protocols will be given.

There exist what is called growing tree based distributed BSF. In this category there are different sorts of *Bluetree* protocols. [21] proposes two such protocols, where the resulting scatternet is called a Bluetree. In these protocols the number of roles each node can assume is limited to two or three. A role here takes the meaning of whether the node is a master or a slave in a piconet. The initiation in the first protocol given by [21] comes from a single node, the Blueroot, which also will be the root of the bluetree. To build a rooted spanning tree the first thing that happens is that the root will be assigned the role of a master node. Now all the one hop neighbors of the root will be its slave. The children of the root is now assigned an additional master role and all their neighbors that are not assigned any roles yet will become slaves of these newly created masters. This procedure will be recursively repeated until all nodes are assigned.

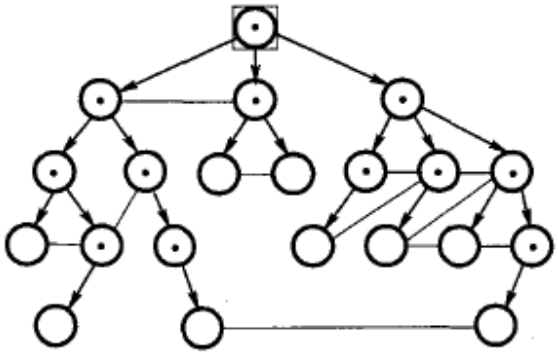


Figure 9: The result of the creation of a bluetree using the Bluetree BSF protocol described above [21]

As can be seen in figure 9 (above) each node is a slave for only one master, which is the first that paged it. Each internal node of the tree is a master in one piconet, and a slave of another master.

In the second protocol suggested by [21] several roots are initially selected, and each of them creates its own scatternet. This protocol has two phases and in the second phase subtree scatternets are connected into one scatternet spanning the entire network.

The main drawback pointed out by [22] with tree based formation protocols is the fact that the topology created has a hierarchical structure. Because of this the created scatternet lacks reliability. If one parent node is lost, all the children and grandchildren nodes below it will be separated from the network and part of or the whole tree has to be rebuilt to retain connectivity. Since nodes are lost quite frequently in mobile networks this makes the Bluetree protocol somewhat vulnerable. In addition there are other drawbacks, such as communication overhead due to maintenance procedures.

In addition there exists other growing tree algorithms, but all of them have drawbacks in common, and that is the above mentioned problem with hierarchical structures in mobile networks and also communication overhead which is can be significant – especially when maintenance procedures are designed and added to the protocol.

In [22] a solution under the *Clustering based BSF* category is presented. As for all protocols drawbacks are found in this protocol category too, namely the fact that the protocol does not always lead to a connected structure. However, on the positive side, two great measures for scatternet performance are proposed; the average shortest-path length and maximum traffic flow.

Context, on-demand and QoS based distributed BSF is the third category of protocols in multihop networks. An example of a Context based BSF is a protocol called BlueScouts, which is an on-demand BSF based on mobile agents [23]. This protocol runs in an asynchronous fashion, and device discovery is decoupled from actual topology formation.

The proposed approach introduces unmatched flexibility by allowing context-aware topology formation. Other kinds of protocols in this category are pure on-demand scatternet formation and QoS based scatternet formation.

The best algorithm among all the methods that do not use position information is (according to [19]) the BlueMesh algorithm which comes under the *Connected out-degree limited scatternets* category. BlueMesh guarantees connectivity and limits the number of slaves in each piconet. The scatternet formation proceeds in iterations. Initially all nodes are undecided. Init-nodes (having the largest weight among immediate undecided neighbors) create piconets in all iterations by choosing at most seven neighbors as slaves, and deleting the remaining edges. Iteration stops when all nodes are decided. All the masters that are created and the slaves not selected for links with slaves from other piconets withdraw from the next iteration. According to [19] the number of iterations grows slowly with number of nodes. Some weaknesses show on some other metrics, especially on the worst-case number of slave roles a node can assume.

In addition there are *Position based connected and degree limited BSF* protocols. Position based BSF schemes require that all neighboring devices are discovered before the scatternet is created. When this is done degree limiting is done along with preserving connectivity (assigning master and slave roles). Degree limitation is done using graph theory, for example using minimum spanning trees (MST's).

3.1.4.3 Choose a scatternet formation protocol

As one can see there are quite a lot of options when it comes to which scatternet formation protocol to use. First of all there are centralized protocols that do not scale well, as opposed to distributed protocols that works better in large networks. There are the complex, quite accurate protocols, like the protocols based on the BlueMesh algorithm. On the other hand there are the less complex ones based on some bluetree algorithm, which unfortunately does not work too well in dynamic environments. As [19] states, a number of protocols are proposed in the literature, in which very few of them satisfy most of the desirable characteristics. Relatively few of these proposed protocols

are actually implemented and compared. Papers that have compared protocols have concluded that it is the Bluetooth device discovery process that is the most time consuming process, hence scatternet formation is a huge task because of the potential large number of devices.

Nevertheless, BSF is a necessary process if one wants to establish mobile multihop ad hoc networks using Bluetooth devices. When the Bluetooth specification adopts a BSF protocol in an effort to make multihop ad hoc networking easier for the average user of a Bluetooth device is not decided, if it ever will be adopted. It could be that adopting a BSF protocol is a wrong move since different BSF protocols are based on different measures; hence the BSF protocol implementation should be based on the individual needs of the different ad hoc networks. It seems as if the question of BSF protocols is quite open, and decisions regarding this issue are bound to be made in the future.

No definitive correct recommendation can be given towards which BSF protocol should be used in ServiceFrame environments. There is the question of how many devices there will be in the network. If there normally are only a few devices in ServiceFrame environments there is perhaps no need for a distributed protocol, giving room for ring topology solutions which are, as mentioned, relatively easy to implement. On the other hand the number of devices could be quite large, and perhaps one should take this into account when implementing a BSF protocol. In this case maybe a BlueMesh protocol is the most promising alternative.

3.1.5 Bluetooth operations and states

The Bluetooth link controller/baseband layer (presented in 3.1.6) is a state machine – which in other words means that it operates like a computer and usually only do one thing at a time. The Bluetooth devices' radio should be cheap, hence receiving and transmitting simultaneously is not possible [16]. Therefore state diagrams can be made, and this is given in the following subchapter. Not everything in the Bluetooth specification (given in [18]) is implemented on real devices – which is indicated in this chapter.

3.1.5.1 Simple state diagram

To be able to find and connect to other devices some functionality must be supported - namely inquiry and paging. Inquiry is the process of discovering other Bluetooth enabled devices and paging is the process of establishing a connection with a Bluetooth device. Because of limitations on processing capabilities the link controller can not, for example, both listen for its page and partake in a piconet at the same time. Therefore multiplexing between these activities must be done.

A state diagram containing a set of states associated with each step of establishing a piconet, and a set of allowable state transitions is given below:

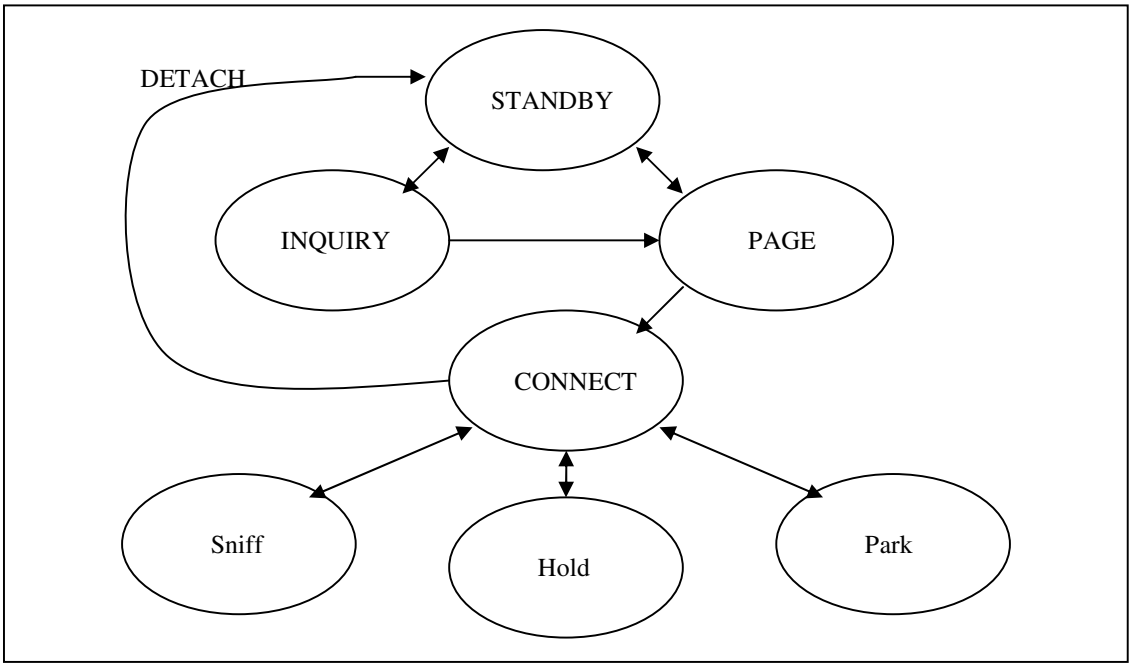


Figure 10: The state diagram showing how to establish a *point-to-point* piconet between one master and one slave (figure 5.2, pp.193 [16])

When a Bluetooth device is powered up it enters the standby state, where hardware and software is initialized. Depending on whether or not the paged units' Bluetooth address (BD_ADDR) is known the device enters either the page or inquiry state. The inquiry state enables discovery of the BD_ADDR of all respondents, from where it can either enter standby again or go to the page state to establish point-to-point connections with the respondents. A successful page results in a master and a slave device, and the devices

starts to exchange network setup parameters – in which they both have entered the connect state.

While a device is in the connect state the slave may arrange with its master to enter a low-power mode; either sniff, hold or park. In the sniff mode the slave checks for a master transmission less often than in every even-numbered slot. The hold mode is a time during which the slave temporarily exits the established piconet without disconnecting from the master. When the hold time is expired the slave can perform normal piconet operations. If a slave enters the park mode it only activates its receiver periodically to resynchronize with its master. To resume communication the master must unpark the slave.

3.1.5.2 A more complex state diagram

The diagram presented before only applies to a *point-to-point* link between a master and one slave. This is because the master in the diagram can not return to the page or inquiry states from the connect state. Using this state diagram no new slaves can be brought into the piconet. Some older chipsets used in mobile phones actually have this limitation. The state diagram shown in figure 11 supports behavior needed for normal piconet and scatternet support. Unfortunately vendors have chosen not to include some of the states described in this subchapter, making scatternet operation impossible.

Having the capabilities of this state diagram a master of a piconet can easily bring a new slave into the piconet. This is done by having the possibility to move from the connect state back into the page or inquiry state.

With this diagram we see that a device enters one of four states after being powered up in the standby state. If a master knows the BD_ADDR of a particular slave it enters the page state to establish a piconet with this slave. Page scan is used by a slave to listen for its page, whereas the inquiry scan is used to listen for an inquiry. The inquiry state is used by a master to discover the BD_ADDR and other information of devices in range.

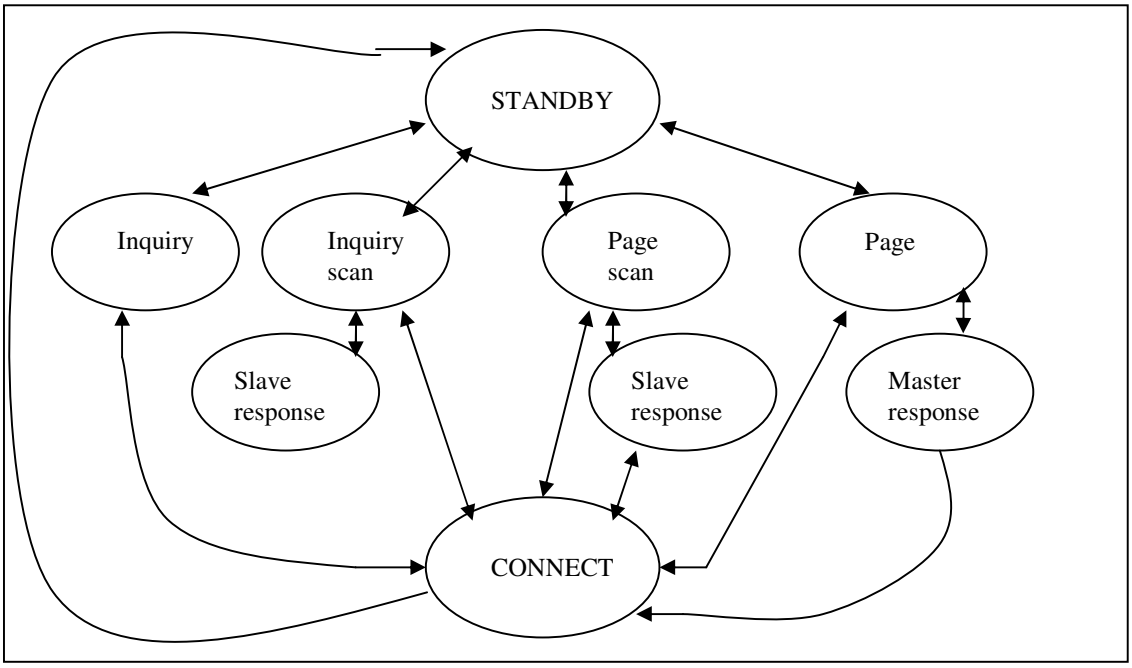


Figure 11: A general state diagram for connecting Bluetooth devices (figure 5.3, pp.195 in [16]). In the diagram the low-power modes sniff, hold and parked are removed. This diagram can be used to establish point-to-multipoint connections.

Three of the states mentioned have responses associated with them. If a slave hears its page while in the page scan state, it responds with its own device access code (DAC) and the slave response state is used. If the master hears the slave's DAC response it responds giving the slave enough information to hop with the frequency of the master during piconet operation. Here the master response state is used. When a slave hears an inquiry while still in the inquiry scan state, it will respond with some information.

Inquiry scan and page scan are the two states that are missing in the Bluetooth chipset implementations used on mobile devices. This makes it impossible to do scatternet operations.

3.1.5.3 Where to put scatternet operations?

A regular Java programmer can not do anything about the lack of scatternet operation. However a programmer with access to the Bluetooth Host stack can enable scatternet support, but as will be outlined this is not recommended.

The scatternet operation support can be put on the Host stack or to the Bluetooth control layers below the Host Controller Interface (HCI). The risks by putting these operations to the Host stack is shown in [34] and hence argues why such operations should be located with the control layers of the Bluetooth implementation. A risk mentioned in [34] is an interoperability issue like the fact that the host stack does not have a complete overview when two master clocks overlap, and hence does not know when to adjust the sniff window causing timing difficulties. To work around this a number of manufacturer specific HCI commands that makes it possible for the host stack to receive the necessary timing information from the baseband layers can be implemented.

The greatest risk according to [34] is the fact that this is a breakdown of interoperability, where seamless use of a Bluetooth host stack from one manufacturer together with a Bluetooth baseband of another manufacturer might not be possible because of the manufacturer special commands.

3.1.6 Protocol stack

For application developers, the Bluetooth protocol can be split up into two main items: layers and profiles. All the layers of the Bluetooth protocol form the protocol stack, which will be described here. Profiles are described in the next subchapter. Both this and the next subchapter are given for the sake of completeness.

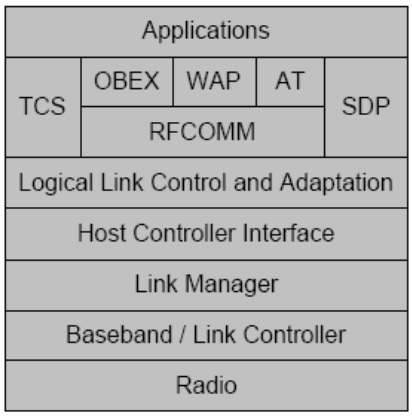


Figure 12: high-level view of the architecture of the Bluetooth protocol stack [24]

More on this and the technical specifications is given in [12] and [13].

The radio layer

This is the physical wireless connection. Fast frequency hopping is used to avoid interference with other devices that also communicate in the ISM (industrial, scientific, and medical) band.

The baseband layer

The layer is responsible for controlling and sending data packets over the radio link. Through this layer transmission channels for both data and voice is provided. According to [34] vendors of Bluetooth chipsets should put support for scatternet operations into this layer.

The link management layer

Here the links set up by the baseband are used to establish connections and to manage piconets. Channel access control is an important task, as well as link-level access control. For this Time Division Multiplexing (TDM) is used, where each time slot is 625 μsecs. The slots toggle between the master and slaves, meaning that when the master uses the slot for transmission, the slave is receiving and cannot transmit. Other tasks for this layer are authentication, security services and monitoring of service quality.

The Host Controller Interface (HCI) layer

The HCI is a layer of software that passes data from the computer to the attached Bluetooth device. If one uses a Bluetooth device attached to the USB port one needs a layer that can understand the USB calls and send that information to the layers above HCI in the stack.

The logical link control and adaptation protocol (L2CAP) layer

This layer is a core layer of the stack. All data must pass this layer, where packet segmentation and reassembling is done. In other words this layer receives and adapts data to the Bluetooth format.

RFCOMM

RFCOMM is known as the wireless serial port, or the cable replacement protocol. The protocol simulates the functionality of a standard serial port. This is the protocol in which the Bluetooth modules of ActorFrame and MIDletActorFrame are based on.

Service Discovery Protocol (SDP)

In order to be able to discover services provided by units which are Bluetooth-enabled Bluetooth uses SDP.

Telephony Control Protocol Specification (TCS)

For devices that want to employ the audio capabilities within Bluetooth TCS is used to send control signals.

Object Exchange (OBEX)

OBEX is a communication protocol, which is quite useful when transfer of objects like files between Bluetooth devices is needed.

Wireless Application Protocol (WAP)

WAP is used on Internet-enabled wireless phones, and in Bluetooth this is an adopted protocol incorporated into the Bluetooth protocol to fit Bluetooth's needs.

AT protocol

This protocol allows the device to be configured to cable replacement mode.

3.1.7 Bluetooth Profiles

To decide whether two Bluetooth-enabled devices can interact properly there are different Bluetooth profiles. For the devices to interact without error both devices must support the profile. In the following the most used and important profiles for this project is described.

Generic Access Profile

This profile is the most common Bluetooth profile, and at a minimum all devices must support it. In the profile connection procedures, device discovery and link management is described.

Service Discovery Application Profile

The definition of features and procedures needed for an application to do service discovery is described in this profile. To do this, direct interaction with the SDP layer in the protocol stack is needed.

Serial Port Profile

The Serial Port Profile interacts directly with the RFCOMM layer in the Bluetooth protocol stack. The profile is used to create a virtual serial port on a Bluetooth-enabled device. This means that one communicate over the virtual serial port as if it actually were a serial port.

In addition there are several other profiles; FAX Profile, Headset Profile, LAN Access Profile, Personal Area Networking Profile, Object Push Profile and File Transfer Profile.

The different profiles are dependent on each other, whereas every profile is dependent on the Generic Access Profile, there exist other dependencies for other profiles. The reason for this is that the Bluetooth profiles where designed to be building blocks, where a higher level profile is dependent upon the functionality of the lower profiles.

3.1.8 Bluetooth limitations

An important issue in respect to this thesis is how routing is performed in networks consisting of several Bluetooth piconets – also known as scatternets. Bluetooth itself actually does not address routing, and this is a great limitation [15]. Which unicast or multicast ad hoc routing schemes to use are not specified in the Bluetooth specification. Actually most network functions are pushed into the link layer. As described in 3.1.4 the Bluetooth standard does not specify a BSF protocol, so this will have to be implemented

to be able to utilize Bluetooth devices in mobile ad hoc networks. Which BSF that is best depends on what purpose the ad hoc network will serve. The BSF protocol can be designed on several different criteria: maximum capacity, minimal average load, minimal average path length, node availability, communication overhead and so on.

In addition to the issue with routing other limitations exist as well. For example there is an upper bound of 8 active devices in a piconet. Another limitation is the device acting as the master. This device will be a limitation because all communication in a piconet goes via the master – which potentially can give trouble with processing power.

Different Bluetooth devices like mobile phones can have different hardware implementation of the Bluetooth specification, meaning they have different Bluetooth chipsets. Among the most important properties that vary among the different chipsets is support for master/slave switching – important for scatternet creation, and if the device can enter the inquiry, page, inquiry-scan, page-scan states described in 3.1.5. These last properties are important for scatternet formation, and hence a requirement to make ad hoc networks. Most devices/chipsets support inquiry and page, but inquiry-scan and page-scan are not as common. [35, 36] give a strong indication that scatternet operation is not implemented on mobile phones. These lacks in implementation make mobile phones unsuited for Bluetooth ad hoc networking at the present time.

3.2 The Java technology - J2ME (Java 2 Micro Edition)

For developing applications for small devices such as pagers, mobile phones and PDAs the Java family can offer J2ME. J2ME was the reaction to the explosion of small-devices that needed Java [14]. In this subchapter an introduction to J2ME is given. To present J2ME a description of the architecture, configuration and profiles is given. In addition MIDP 2.0 will be elaborated to some extent. The MIDletActorFrame package is based on J2ME, and this is why an elaboration of this technology is important.

3.2.1 The architecture

According to [25] the J2ME architecture defines configurations, profiles and optional packages. This provides specific information about APIs and different families of devices. An overview of this architecture is also found it [25].

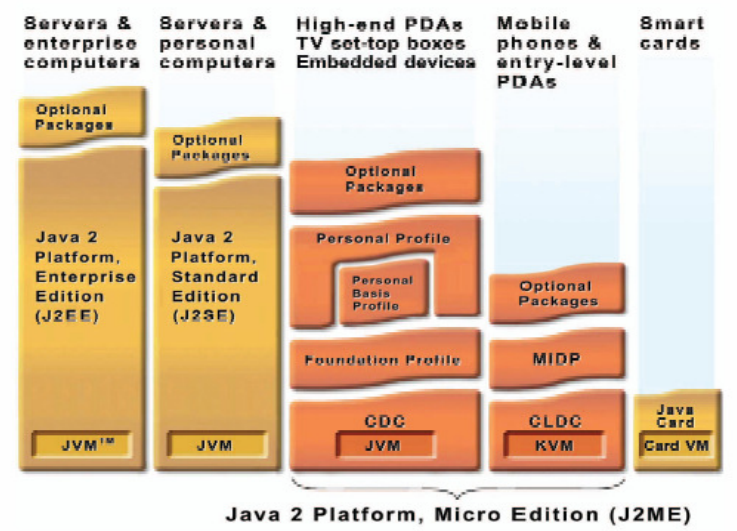


Figure 13: An overview of the J2ME architecture [25]

The configurations

A configuration specifies a Java Virtual Machine (JVM) (see [26] for elaboration on this) and some set of core APIs for a specific family of devices. There are currently two configurations; the Connected Device Configuration (CDC) and the Connected, Limited Device Configuration (CLDC) [25], [14].

I. Connected Device Configuration

At a minimum, a connected device has 512KB of read-only memory (ROM), 256KB of random access memory (RAM), and also a network connection. CDC is designed for devices like TV set-top boxes, car navigation systems and high end PDAs. For CDC it is specified that a full JVM must be supported. Because of this, the configuration is targeted for devices with a minimum of 2 MB of memory available for the Java platform.

II. Connected, Limited Device Configuration

More interesting for this thesis is the CLDC, because it encompasses mobile phones, pagers, PDAs, and other devices of the similar size. CLDC is designed for devices with 160 KB - 520 KB of memory available for the Java platform. The “limited” word in the

name points to the fact that the devices have limited network connections which are intermittent and not very fast.

CLDC is not specified for a full JVM implementation. It is based around a small JVM called KVM, where the name comes from the fact that it is a JVM whose size is measured in kilobytes rather than megabytes (K = Kilo). The configuration comes in different versions, with CLDC 1.0 containing less functionality than CLDC 1.1, which is more adapted with for example support for floating points important for doing mathematical calculations.

3.2.2 The profiles

A profile is layered on top of a configuration, adding higher level APIs in order to provide a complete runtime environment targeted at specific devices. There are several different profiles available today: Mobile Information Device Profile (MIDP), Foundation Profile (FP), Personal Profile (PP) and Personal Basis Profile (PBP).

Mobile Information Device Profile

The profile has to this date come in two versions, MIDP 1.0 and MIDP 2.0. MIDletActorFrame runs on MIDP 2.0, thus the focus will be on this version. According to the specifications, MIDP has the following characteristics [14]:

- 128KB of non-volatile memory for the MIDP implementation
- 32KB of volatile memory for the runtime heap
- 8KB of non-volatile memory for persistent data
- A screen of at least 96 x 54 pixels
- Some capacity for input, either by keypad, keyboard, or touch screen
- Two-way network connection, possibly intermittent

A MIDP application is called a MIDlet, and has APIs from both CLDC and MIDP available [14]:

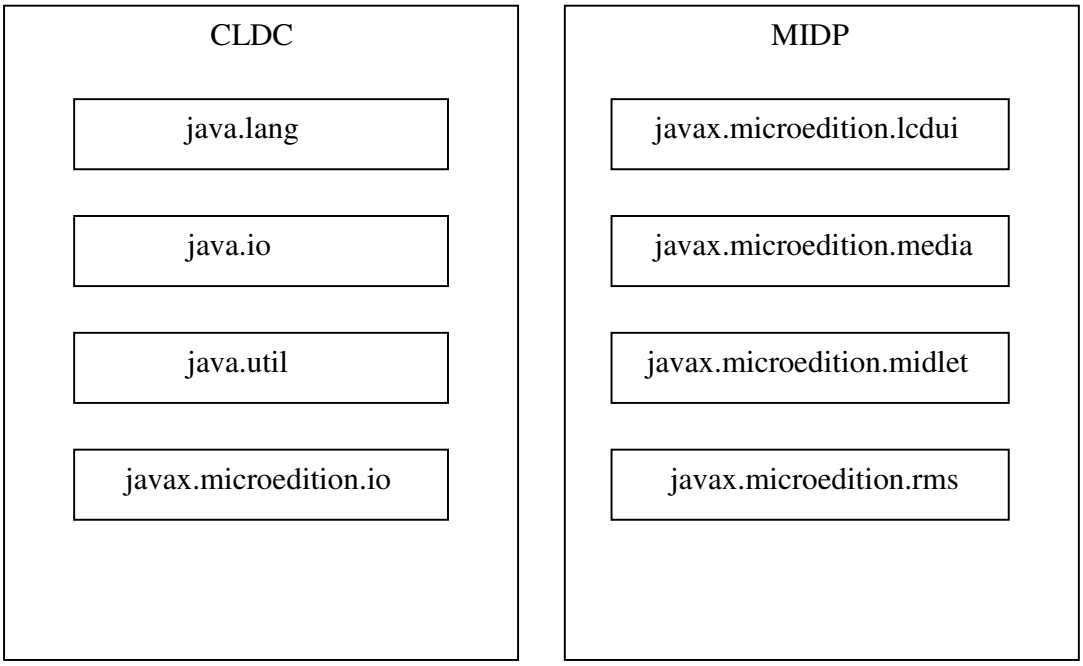


Figure 14: The packages a MIDlet have access to [14]

From this figure we see that CLDC defines a core mostly taken from J2SE (Java 2 Standard Edition).

An important aspect of MIDlets is their life cycle. The life cycle is reflected in the methods and behavior of the MIDlet class. A MIDlet is installed by moving its class files to a device, where the class files will be packaged in a Java Archive (JAR) with a .jar extension. Along with this archive a descriptor file (with a .jad extension) follows, which describes the contents of the archive file. The JAD file actually allows the application management software (AMS) on the device to identify what it is installing prior to actually installing the MIDlet. Instead of having a main method, the AMS which is part of the operating environment, execute methods allowing the MIDlet to enter different states [27].

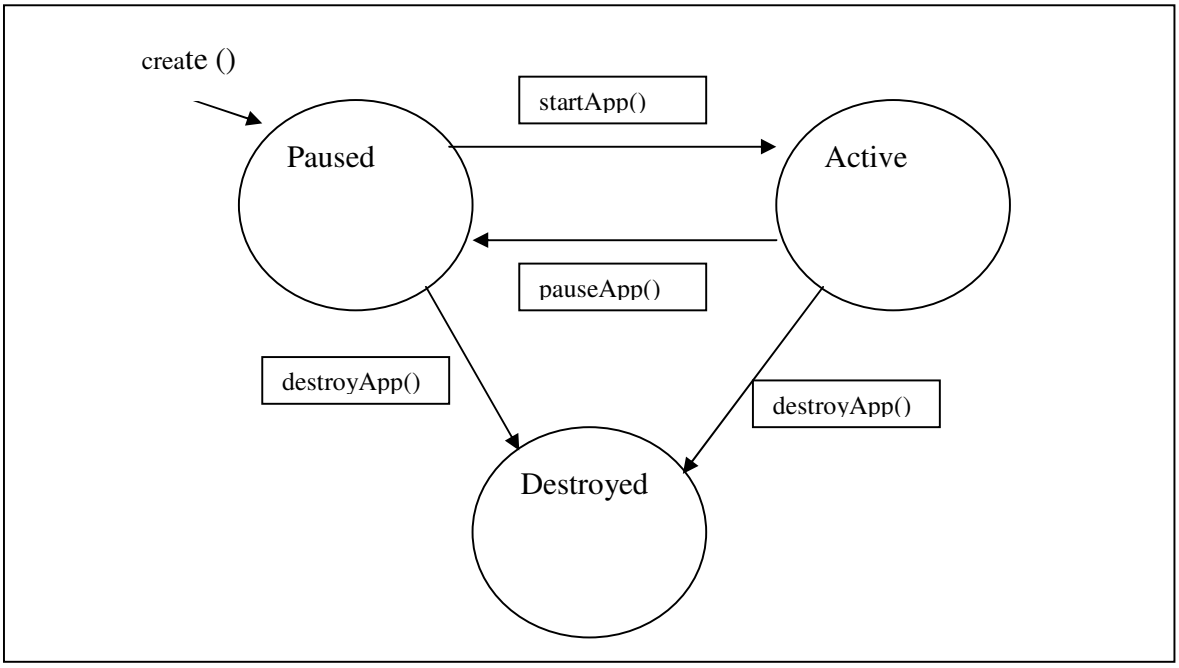


Figure 15: Illustration of the states of a MIDlet and the transitions between them [27]

There also exist one additional method in the MIDlet class, namely resumerequest (), which can be used from a paused state to signal to the AMS that the MIDlet wants to become Active.

3.2.3 Record Management System (RMS)

Since MIDP applications have to run seamlessly on many devices there are some challenges, for example in user-interfaces and persistent storage. To solve the challenges concerning storage, MIDlets do not care about what kind of storage it is run on. The MIDlet only care about small databases called record stores. In the record stores pieces of data called records are present. This storage system is called Record Management System [14]. In MIDP 2.0 a MIDlet can access record stores created by other MIDlets, which was not possible with MIDP 1.0.

In detail a record store consists of records, where each record is a byte of data of varying size depending on the data stored. This is shown in table 1:

Table 1: The record store database

int id	byte[] data
1	[data]
2	[data]
3	[data]
etc.	etc.

As can be seen each record in a record store has a unique integer identification number starting the numbering at 1. When a record is deleted the identifier will not be reused. To get access to the record store the RecordStore API provides methods for adding, deleting and changing records.

3.2.4 More about the optional packages (APIs)

As one could see in the overview of the J2ME architecture in 3.2.1, there exist optional packages, or optional APIs. The most relevant optional API for this project is the Java Bluetooth API (JSR82) and MMAPI (JSR135). MMAPI is mainly explained in 5.1.1 JSR82 is an additional API that consists of two packages; the Bluetooth API (javax.bluetooth) and the OBEX API (javax.obex). These APIs do not implement the Bluetooth specification, but provide a set of APIs to access and control a Bluetooth-enabled device [13].

3.2.4.1 JSR 82 – The Bluetooth API

The main capabilities that JSR82 is intended to provide are according to [13]:

- Register services
- Discover devices and services
- Establish RFCOMM, L2CAP, and OBEX connections between devices
- Using those connections, send and receive data (voice communication not supported)
- Manage and control the communication connections
- Provide security for these activities

To be able to use JSR 82, it is required that the Bluetooth stack underlying the JSR 82 implementation is qualified for the Generic Access Profile, the Service Discovery Application Profile, and the Serial Port Profile. These profiles are mentioned in 3.1.7 as part of the Bluetooth protocol specification. The stack must also provide access to its Service Discovery Protocol, and to the RFCOMM and L2CAP layers.

Bluetooth Scatternet Formation protocols depend on the devices they are implemented on to support switches between master and slave. For the BSF algorithms to work a device must be able to dynamically change its role from a slave to a master or vice versa, if it is needed. Old mobile phones that have JSR82 support (like SE P900) does not support master/slave switching. Sony Ericsson phones with JSR82 support announced later than the first quarter of 2005, like the K750, are suspected to have support for master/slave switching according to [28]. As indicated in 3.1.8 this is not enough for scatternet operation. The source of this information ([28]) only shows how difficult it is to find information about what the different devices support. Eventually all the specifications in [18] will probably be implemented, but until that is guaranteed it is very difficult to say what the different mobile devices support.

3.2.4.2 JSR 259 – Ad Hoc Networking API

The optional API JSR 259 is a Java Community Process (JCP) that was initiated by Siemens (Ben-Q) together with some other vendors (Nokia, Panasonic). This API is mentioned here because it is an API that will provide a lot of the functionality discussed in this thesis. The scope of JSR 259 is to provide a generic mobile ad hoc communication mechanism between nodes in an ad hoc network implemented in J2ME, which hide the actual architecture and complexity from the developers [42].

Concrete ad hoc networking implementation and mechanisms will not be defined, but it will be generic enough to support different implementations. Some of the supported methods that JSR 259 will support are:

- Service Discovery
- Service Registration

- Service Availability Alert
- Service Consumption
- Service and Service Capability Inquiry

Through this optional API third party vendors can make P2P applications for mobile phones. This means that JSR 259 has some overlapping goals with ServiceFrame, which already has implemented a framework for most the functionality mentioned above.

The first draft of JSR 259 came in March 2006 and hence there will be some time before a commercially available version of this optional package is available.

4 Routing

In this thesis the evaluation of the existing networking modules in ActorFrame is important, and in focus is the issue of whether the routing protocol used in ActorFrame is suited for use in multihop ad-hoc networks – a term used several times already in this thesis without being properly explained, hence it will be explained in chapter 4.3.2. Basic theory about routing in general and specific routing protocols both for wired networks and wireless ad-hoc networks will be given to support evaluation of the ActorFrame routing protocol.

The routing protocol in ActorFrame exchanges information about available services, and therefore is a routing protocol between applications. Although the routing protocols described in 4.2 are most commonly used in Internet IP routing, the principles can be used for routing between applications – as the ActorFrame routing protocol described in 4.4 will show. Routing between applications is relatively new. Another technology that provides routing between applications is the Simple Object Access Protocol (SOAP), a protocol that allows communication between applications using HTTP. Routing between applications in wireless multihop ad hoc networks is an even less treated subject.

4.1 Routing - two different operations

In general one can say that routing is split in two different operations. One of the operations is *forwarding*, which is the passing of logically addressed packets or messages from their source toward their ultimate destination through intermediate nodes, called routers or gateways [29, 30]. The routers or gateways looks at the destination address in the packet/message header and from this and its routing table/forward table the route or next hop towards the destination is determined, and accordingly forwards the packet/message.

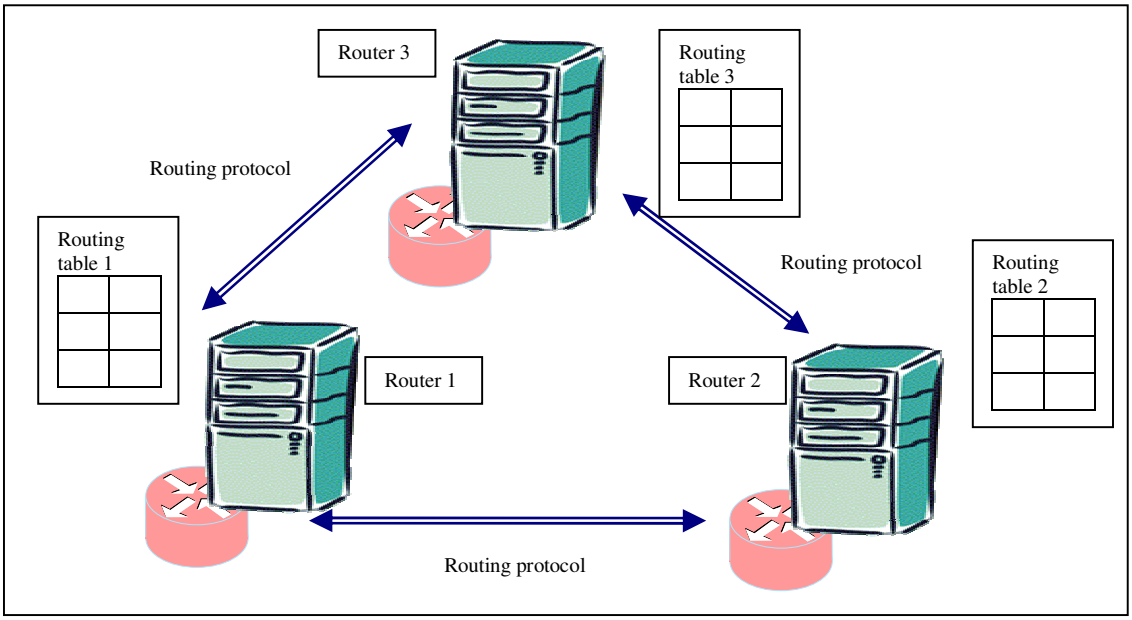


Figure 16: Diagram showing that routers have a routing table updated by information carried by routing protocols.

Routing tables contain information to forward packets to their next hops. The operation that builds and updates the routing tables is called *routing*. How the routing tables are constructed is therefore of great importance for efficient forwarding of packets/messages. Manual construction of routing tables (also known as static routing) is possible if the network is small, but for larger networks manual construction is impractical. The reason for this is that large networks may involve complex topologies and constant changes, making it very hard to keep the routing table updated.

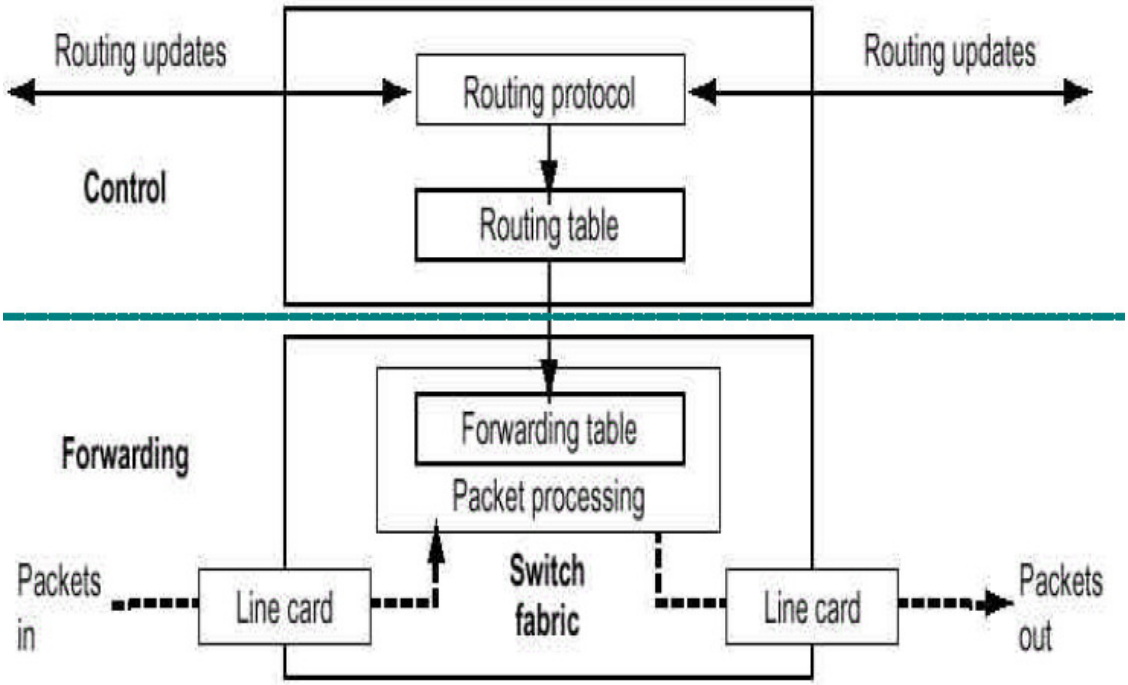


Figure 17: Illustration that shows the separation of routing and forwarding. Packets enter the router on an interface and the destination address is found and compared to known addresses in the forwarding table. If a match is found the packet is forwarded on the correct interface. On top of this the routing protocol ensures that the forwarding table is kept up to date at all times [31]

The alternative to static routing is dynamic routing, in which the routers/gateways are responsible for automatically creating and maintaining the routing table entries. Information required for the automated approach is carried by *routing protocols*. To make it possible for routers to have an updated routing table that represents the ever-changing network topology routing information must be gathered. This routing information is exchanged between routers by means of a routing protocol.
Since routing in the Internet is dominated by the dynamic approach, this will be the focus in this text.

4.2 Routing protocols for wired networks

Wired networks can consist of many different devices interconnected through wire lines. By using wire lines transmission rates can be quite high. Depending on the various wire lines used as transmission medium (fiber, copper, coax or twisted pair) the transmission

rates and available bandwidth vary. For fiber transmission rates of several gigabits per second (Gb/s) is possible, and for the other transmission mediums the rates are lower, but still relatively high. A common characteristic for wired networks no matter what transmission medium is used is that there are no or very infrequent topology changes. This is something that is reflected in the routing protocols constructed for wired networks. Other metrics that routing protocols are adapted to could be for example bandwidth, delay, traffic load and reliability.

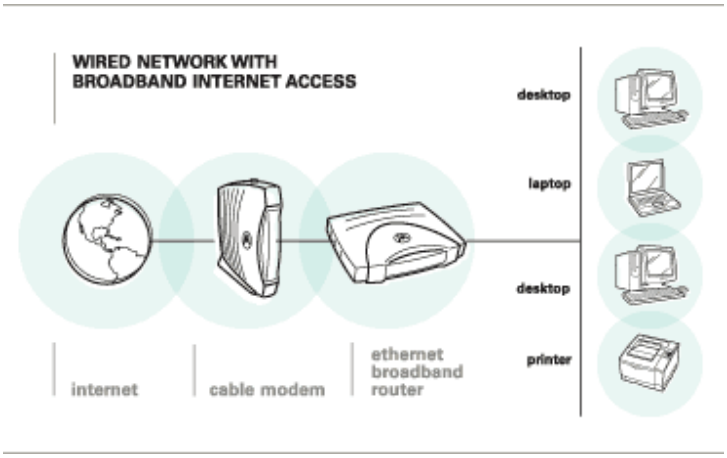


Figure 18: A diagram showing an example of a wired network connecting different devices to the Internet.

To distribute routing information around a network, a complete topology of routing protocol “connections” between neighboring routers in the network must be established. This topology often reflects the topology of the actual physical links between the routers, but this is not necessarily true. To make it simpler and less complex to distribute information and calculate routing tables, dividing large complex networks into smaller areas, subnetworks or domains and to create a hierarchy among the different routers is customary. This way specialized routing protocols can be used on the different network hierarchies, dealing with the different problems that arise on the different levels.

The rest of this subchapter will concentrate on presenting different Internal Gateway Protocols (IGP); routing protocols used to exchange routing information between routers in the Internet. The reason for concentrating on these routing protocols is that ActorFrames’ routing protocol is based on principles found in these protocols. There are

two categories which the routing protocols can be split into – according to whether they are *distance vector protocols* (DVP) or *link state protocols* (LSP). Differences between the two types of protocol categories is the way routing information is stored and advertised, and also in the way routes are calculated.

4.2.1 Distance vector protocols (DVPs)

A DVP works by calculating distance and direction from every possible source router to every possible destination. When the distance and vector is determined it can be advertised as routing information to neighboring routers within the network – so that these routers can calculate their routing tables. The process is iterative, and continues until the routes in the routing domain stabilize [30].

There are two main advantages of DVPs. First of all the algorithms required to calculate routing tables from the received routing information are simple. For example a simple hop count algorithm can be used as the routing algorithm (see figure 19). Second there is a low amount of processing effort and data storage involved in creating routing information in preparation for advertising, where the router actually just advertises the contents of its routing table.

Put in another way – DVPs are not demanding in their use of router processor power or data storage capacity. The drawback is that they are crude in their derivation of routes and each router has to keep on advertising its entire routing table periodically, which can make a considerable additional traffic load for the network. Also it may take some time before the best routes to be made known to all.

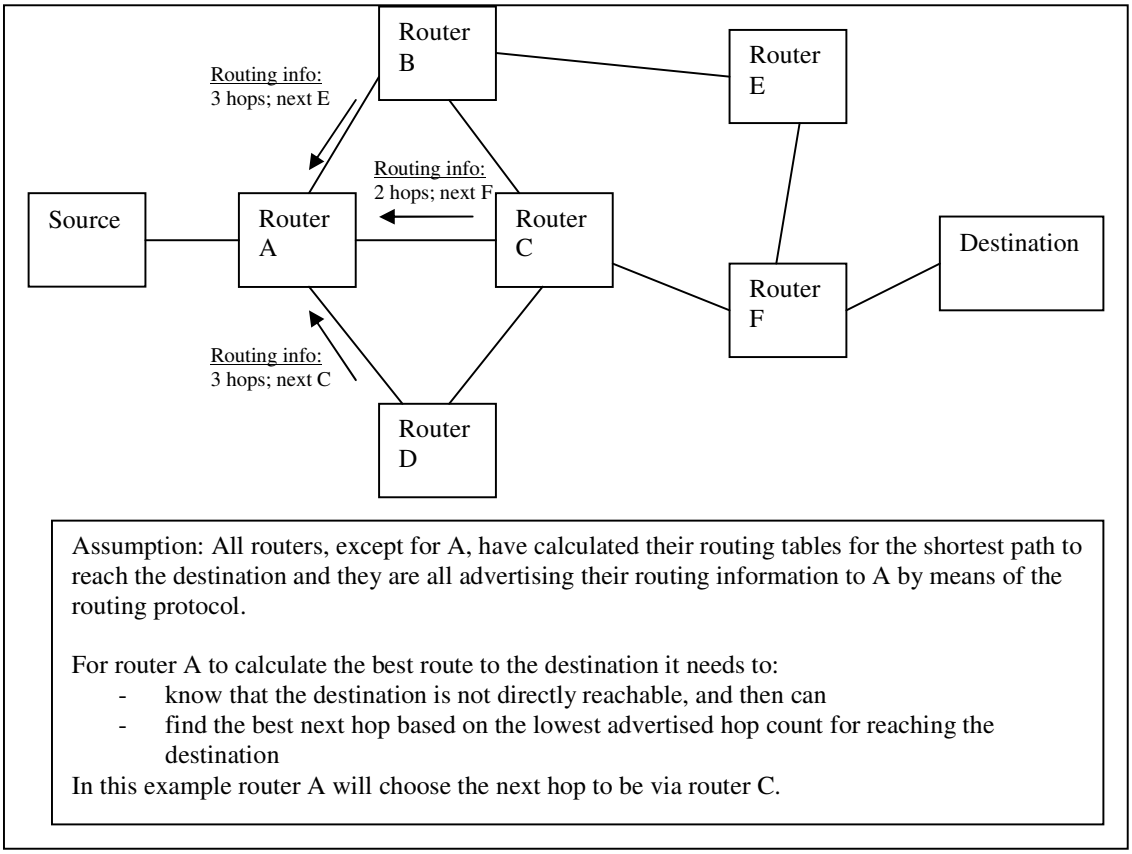


Figure 19: Illustration of how the routing information in DVP can be used to make a decision of the best next hop. (fig. 6.8, page 227 [29])

Examples of DVPs are the Routing Information Protocol (RIP), Cisco's interior gateway routing protocol (IGRP) and BGP (Border Gateway Protocol). An elaboration of RIP will be given in chapter 4.2.3.1.

4.2.2 Link-state protocols (LSPs)

LSPs operate by distributing routing information about the state of the individual links of the network. A router is able to build a complete network topology database of the topology of the network by listening to the routing protocol advertisements of other routers, which are broadcast or flooded to all routers throughout the network. Using this topology database all routers can work out their shortest paths to every individual destination – making it possible to build an optimal routing table.

For LSPs there are three main advantages over DSPs. To start with the quality and efficiency of routes selected across a network is better. There is also better ability to provide for reliable routing even in very large, complex networks. The third advantage is that the amount of routing information (link state advertisements (LSAs)) which has to be advertised by each router is greatly reduced.

Of course there are drawbacks to LSPs too. The routers, for example, must be equipped with large amounts of data storage capacity to store the entire network topology database. Another requirement is the high power processing capability to manage the complex mathematics associated with deriving the shortest path to every possible destination.

Examples of LSPs are IS-IS (intermediate system-intermediate system) and OSPF (Open Shortest Path First). In the next subchapter both a DVP and an LSP is presented, namely RIP and OSPF. This is to give a better understanding of the two types of routing protocols.

4.2.3 Specific routing protocols: RIP and OSPF

Many different routing protocols have been developed and used during the lifetime of the Internet. Two of the most important and most used protocols are RIP and OSPF. In the two following subchapters the two protocols will be presented.

4.2.3.1 Routing Information Protocol (RIP)

In addition to being one of the oldest routing protocols, RIP is also one of the simplest interior gateway protocols. It is a DVP and works by determining the shortest path distance from router to destination as measured in terms of the hop count. RIP runs over UDP (User Datagram Protocol), so its messages (routing information) are encapsulated into UDP segments.

RIP exists in two versions, which both still are in use. Version 2 extends version 1 to cope with classless inter-domain routing (CIDR) and variable length subnet masks. Most importantly RIPv2 includes the option for authentication of routing protocol messages. Authentication prevents “untrusted” third parties from manipulating network routing.

The basics of RIP

The calculation of routing tables and advertising of routing tables in RIP is done the way explained in figure 19 in chapter 4.2.1. This means that all the neighbors of a given router advertise their entire routing table to the router, giving information about which destinations that are reachable and how many hops need to be transited in order to reach them. Then the router chooses the shortest route of those advertised to it and sets it as its preferred route to the given destination and sets its own next hop choice accordingly. This router then advertises this route as part of its routing table to all its neighbors.

By making all routers share their entire routing table with its neighbors it is sufficient to enable all routers' routing tables to be developed. In addition RIP has guards against circular routing and network instability (route flapping).

To ensure that routes and destinations that are withdrawn are removed from routing tables, RIP uses the principle of ageing. This means that even if there have been no changes in the routing table, routers using RIP must advertise their entire routing table to all neighbors once every 30 seconds. This is done to prevent correct routing table entries from being deleted. If no update of a route currently appearing in the routing table is received within a period of 180 seconds the routing table entries are marked invalid. After 240 seconds (additional 60 seconds) the route is deleted if no update has been received in the meantime.

Triggered updates are used when topology changes occur between two 30 second periods. This means that an update is advertised because it is triggered by a routing change at the router sending it.

4.2.3.2 Open Shortest Path First (OSPF)

OSPF is a link state routing protocol, and like RIP it also comes in two versions – OSPFv1 and OSPFv2. From the name one can derive that it is based on the shortest path first (SPF) routing algorithm, also known as the Dijkstra algorithm. The word *open* reflects the fact that the standard is an open standard rather than a proprietary standard.

The outline of OSPF will not be comprehensive. It will be concentrating on the most important aspects, like how routing information is distributed.

The basics of OSPF

The maintenance of a link state database is the most important task done by the routers employing link state routing protocols. In this database information about the network as a whole and the link cost (the "state") of all links, is stored. Using the database each router can separately calculate a shortest path tree with itself as the root. The shortest path tree is calculated using the Dijkstra algorithm. Doing this calculation the routers can determine its routing table by working out the shortest path route to every destination.

Routing information updates in link state protocols is realized through link state advertisements (LSAs). Each router restricts the origination of routing information which it floods to all other OSPF routers to its router-LSA. Here only information on directly connected routers, networks and hosts is contained.

OSPF relies on each router maintaining an up-to-date copy of the link state database. Any changes in network topology must be updated to all other routers by means of link state updates (LSUs) containing LSAs. The process by which this is done is called flooding. This flooding differs from normal multicasting in that OSPF packets are processed by each successive router before being flooded to all adjacent routers. Flooding ensures that each LSA in each update is acknowledged by each router. It is therefore a somewhat more secure way of keeping all link state databases synchronized than using multicasting. The flooding process is more accurately referred to as "reliable flooding". [32]

If one collects all LSAs the complete link state database can be created. This link state database is collected by all the routers using OSPF, giving them identical link state databases.

Benefits of OSPF over RIP [29]

As mentioned in 4.2.3.1 OSPF is the "IGP of choice". Some of the reasons for this are given below.

The calculations of the shortest path in OSPF are based on link cost. Whereas RIP only depends on one parameter, namely hop count, OSPF is more sophisticated and the link cost can be weighted to accommodate several other metrics such as link bandwidth, delay, load and reliability. This is a benefit because it gives routing schemes that not only adapt to routing topology changes, but also according to network operational conditions.

Protocol	RIP	OSPF
Type	distance-vector	link-state
Convergence Time	slow	fast
VLSM	no	yes
Bandwidth Consumption	high	low
Resource Consumption	low	high
Multi-path Support	no	yes
Scales Well	no	yes
Proprietary	no	no
Routers Non-IP Protocols	no	no

Table 2: A table summing up some of the differences between RIP and OSPF [31]

Network traffic load is reduced using a link state routing protocol and also by limiting routing information messages by only sending "real" update messages. This means that routing information is only sent on updates. One exception is when no topology change is registered in 30 minutes – then a timer will trigger sending of LSA. Further reduction in routing protocol traffic is achieved through subdivision of OSPF routing domains into separate routing areas providing hierarchical routing.

Other functionality giving benefits are load sharing of traffic between alternative paths and that OSPF traffic always is authenticated, so that only trusted routers can take part in the routing process. As opposed to RIP which is a rather slow converging protocol, OSPF is a fast converging protocol requiring minimum routing message traffic. The drawback is that OSPF demands much more router processing power and memory capacity. More processing is needed because the complexity of the calculations necessary to determine the shortest path tree is very complex. Memory capacity must be quite large because in addition to the routing table the routers must store the link state database.

4.3 Routing protocols for wireless networks

So far this chapter has given an outline on routing protocols only valid for wired networks. Bluetooth, however, is a wireless communication protocol and has together with other wireless communication protocols characteristics that are different from wired communication. There exist different kinds of wireless communication, both based on infrared and radio frequency signals. Among these are Bluetooth, infrared transmission, Hiperlan and the 802.11 protocols, which are also called Wireless Local Area Network protocols (WLANs) [33].

4.3.1 Characteristics of wireless communication

Wireless networking is generally speaking the use of infrared or radio frequency signals to share information and resources between devices. Examples of such wireless devices could be mobile terminals, laptops, mobile phones, Personal Digital Assistants (PDA's) or wireless sensors [33].

There are differences in the characteristics of wireless and wired communication. First of all there is usually lower bandwidth availability and much lower power transmission rates in wireless communication. The bandwidth is usually so much lower that the slow-speed introduces additional jitter, delays and longer connection setup times. The network conditions are typically highly variable. For example data loss is higher due to the high interference introduced by wireless communication. Wireless transmission has a broadcast nature and therefore all devices are potentially interfering with each other. Radio signals can be interfered by other electrical devices operating in the same

frequency band, like for example Bluetooth enabled devices and microwave ovens (see chapter 3.1.2). The data loss introduced by interference causes wireless networks to be less reliable compared to wired networks. Another factor that causes variable network conditions is the fact that users are mobile and can potentially be on the move at all times, which cause frequent connect and disconnect operations.

Devices used in wireless networks have limited computing and energy resources, which gives them limitations on computing power, battery capacity and also device size, weight and cost.

Another important aspect is security. Although security is not an issue in this thesis it is well worth mentioning that wireless communication has weaker security. The transmissions are not limited to the confines of a cable [16]. Implementing network security in wireless networks is more difficult than in wired environments where transmission interception is harder because access to the wire lines is required, as opposed to wireless communication which can be intercepted by “everyone” with the “right” equipment.

4.3.2 Mobile ad-hoc networks (MANETs)

There are basically two types of wireless networks. Infrastructure-based networks are networks with preconstructed infrastructure made of fixed and wired network nodes and gateways. As an example mobile phone networks are of this type because they are built from Public Switched Telephone Network (PSTN) backbone switches, Mobile Switching Centers (MSC), base stations and mobile hosts. Also WLANs fall into this category.

The counterpart of such a network type is the infrastructureless (ad hoc) network, illustrated by figure 20. This type of network is formed dynamically through collaboration of an arbitrary set of independent nodes.

Figure 20: An illustration of a small ad-hoc network. Worth noting is the fact that there is no predefined infrastructure. Because of this nodes are expected to behave as routers and take part in discovery and maintenance of routes to other nodes.

As opposed to the first wireless network type mentioned, the nodes have no prearranged specific role it should assume (like the MSC or the base station). Each node makes its decision independently, based on the network situation – not taking into consideration preexisting network infrastructure.

Ad hoc wireless networks are self-organizing and adaptive [15]. In other words a formed network can be formed on-the-fly without any system administration. “Ad-hoc” actually tends to imply “can take different forms” and “can be mobile, standalone or networked”.

4.3.3 The routing protocols

In wireless networks mobility is a dominating factor and this implies that links are tore down and set up again quite often [15]. Since mobile phones and other small, portable and highly integrated mobile devices exist, the nodes in an ad hoc network can move very freely, which results in a dynamically changing topology. The distance-vector and link-state based routing protocols presented in chapter 4.2 are not able to catch up with the frequent link changes in ad hoc wireless networks. Using these protocols would result in poor route convergence and very low communication throughput. This is why other protocols are needed in wireless networks.

There are basically two different kinds of ad hoc routing protocols. The proactive, table-driven, approach attempts to maintain consistent, up-to-date routing information from each node to every other node in the network. This approach require the nodes to maintain one or more tables to store routing information, and changes in network topology is responded by propagating route updates throughout the network to maintain a consistent network view.

The second approach is the reactive (on-demand) protocols. These protocols only create routes when the source has something to send. When a node need a route to a given destination a route discovery process is initiated within the network. After a route is discovered and established, it is maintained by some route maintenance procedure until the destination is unreachable from the source or the route is no longer needed. There are a great number of different protocols:

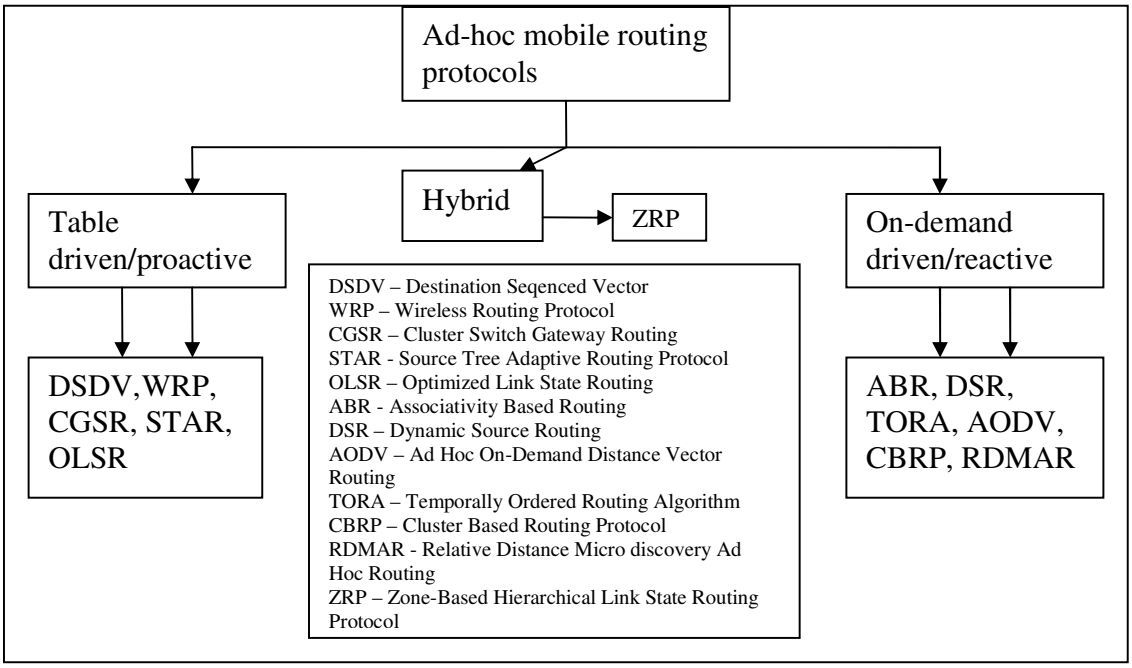


Figure 21: Categorization of ad hoc routing protocols

In the next subchapters some of the protocols will be given some extra attention.

4.3.3.1 Proactive protocols

The Destination-Sequenced Distance-Vector (DSDV) protocol is a distance vector protocol with extensions making it more suitable for MANETs [33]. All nodes maintain a routing table with one route entry for each destination (shortest path route recorded), and routing table updates are sent periodically. Loops are avoided by using sequence numbers. Cluster Switch Gateway Routing (CGSR) extends DSDV with a cluster framework concept improving scalability. The Wireless Routing Protocol (WRP) is another loop-free proactive protocol which uses four tables to maintain distance, link cost, routes, and message retransmission information. Route updates are sent among neighboring nodes with distance and second-to-last hop information resulting in faster convergence.

Although DSDV, CGSR and WRP have their differences, like the number of routing tables and the different routing information kept in these tables – all the protocols have the same degree of complexity during link failures and additions.

The Optimized Link State Routing (OLSR) protocol does as one can suspect from the name; it optimizes the Link State protocol, which was mentioned in 4.2.3.2. In OLSR the node's complete link state information is not flooded to all nodes in the network. Only link information from a subset of links to the neighboring Multipoint Relay Selectors (MRS) is flooded to other MRSs which are responsible for flooding the information to its neighbor nodes. Doing this reduces routing overhead and improves bandwidth efficiency.

4.3.3.2 Reactive protocols

Dynamic Source Routing (DSR) is a loop free on-demand routing protocol, where every node maintain route caches that contain the source routes learned by the node and the route discovery process is only initiated when a source node does not have a valid route to the destination in its route cache. Entries in the cache are continually updated as new routes are learned. In Bluetooth one must discover and connect to all neighbor nodes to be able to flood route requests. Therefore routing protocols must be specialized to work optimally in a Bluetooth environment.

The Ad-hoc On-Demand Distance Vector (AODV) protocol improves DSDV. The number of route broadcasts are minimized in AODV by creating routes on an on-demand basis, versus maintaining a complete list of routes as in DSDV. Route discovery is, as in DSR, on-demand based. The route request is forwarded to the neighbors, and so on, until either the destination node or an intermediate node with a fresh route to the destination node is located. Each DSR packet must contain the full routing path information and therefore has potentially larger control overhead and memory requirements than AODV. AODV packets only contain the destination address. On the other hand DSR works with both asymmetric and symmetric links during routing, where AODV only work with symmetric links. Also DSR maintain multiple routes to a destination in the cache, which is helpful during link failures.

Both DSR and AODV work well in small-to-medium-sized networks with moderate mobility.

4.3.3.3 Hybrid protocols

Hybrid protocols utilize aspects from both proactive and reactive protocols. The Zone-Based Hierarchical Link State Routing Protocol (ZRP) divides the network into nonoverlapping zones based on nodes' geolocation. Neighboring zone connectivity information is propagated by dedicated gateway nodes. Internally in the local zones proactive protocols are used, while between zones reactive routing protocols are used. Such an approach is bandwidth efficient and scalable [33], but the extra cost to maintain the structure can be prohibitive, especially in mobile ad hoc environments – where constant changes can result in unstable structures.

4.3.3.4 Proactive protocols versus Reactive protocols

Generally speaking, on-demand reactive protocols are more efficient than proactive routing protocols in terms of control overhead and power consumption since routes are only established when required [33]. Proactive protocols require periodic route updates to be able to keep information current and consistent. Many routes that might never be needed are maintained – adding quite a significant routing overhead in a bandwidth constrained network, which Bluetooth certainly is. Actually routing overhead grows

exponentially with network size and prevents application of proactive protocols in large-scaled networks.

If one considers quality of service the proactive routing protocols generally are better than on-demand protocols [33]. In proactive protocols routing information is constantly updated and hence routes to every destination are always available and up-to-date, which in turn can minimize the end-to-end delay. The source has to wait for the route discovery in on-demand protocols, and this latency might be intolerable for real-time communication.

Since streaming of media, for example, not actually is real-time *communication,* it seems as if an on-demand routing protocol would be the most efficient routing protocol in the Bluetooth/ServiceFrame environment that this thesis deals with. Since Bluetooth have a relatively limited theoretical bandwidth of from about 700 kbps to 3 Mbps depending on which version of Bluetooth is used, the proactive protocols might imply too much control overhead. If too much overhead is present there might be little bandwidth left for the actual information sent between applications, which potentially could demand quite a lot of bandwidth to work optimally.

On the other hand, [33] claims that proactive protocols are sufficient for a small-scale static networks and reactive protocols work well for medium sized networks. So one should perhaps also take into consideration the size of the network in which it is realistic that Bluetooth enabled ServiceFrame based devices are present. Assumed that the ServiceFrame based devices are only used inside a house there is a limitation on how many Bluetooth nodes that will be present, and hence the proactive approach might be applicable, as there is a limited amount of routing information in small networks. One can in addition to using the ServiceFrame based device at home in quite limited surroundings also imagine that such a device can be used in more public areas where many people have their own ServiceFrame based device activated. Now the proactive approach has its shortcomings and a reactive approach might be preferable.

Another issue pointed out by [41] are the limitations of Bluetooth compared to for example 802.11 (WLAN). Bluetooth is a connection oriented protocol, so to be able to send data to another Bluetooth device a connection must be set up. As mentioned in 3.1.8 Bluetooth has no broadcast capability, so to flood a route request (like in a reactive protocol) one need to connect to all the neighbors first. The inquiry time and connection set up times in Bluetooth are very long so any routing protocol used in a Bluetooth network must take this into consideration. To overcome the limitations given by Bluetooth [41] gives a routing protocol suggestion called Bluetooth Scatternet Routing (BSR). This routing protocol is a reactive protocol like AODV and DSR, but to avoid long delays due to long connection set up and inquiry additional information on the state of the links are kept.

No matter what protocol one chooses to implement in the ServiceFrame/ActorFrame environment one has to take into consideration the shortcomings of Bluetooth. That means that both the reactive and proactive routing protocols can be used, but they must most likely be adapted for use in a Bluetooth environment. Nevertheless there are, as outlined, pros and cons with both proactive and reactive routing protocols.

4.4 The ActorFrame routing protocol

The Bluetooth functionality integrated only recently to ActorFrame changed the routing architecture in the package made for small mobile devices (MIDletActorFrame, chapter 2.2). This change was done so that also these small devices could act as routers and mediators. However, the routing protocol was not changed, so no changes have been made in the routing protocol despite the introduction of Bluetooth, which is a wireless protocol. There are some small differences when it comes to the timers used in the ActorFrame and MIDletActorFrame package, besides that the protocols are very much the same. As explained earlier in this chapter routing is actually two separate operations. The forwarding operation in ActorFrame, which is not a part of building and maintaining the forwarding table, is explained in more detail in chapter 2.1.2.3. Building and maintaining the forwarding table is done by the routing protocol explained in this chapter.

4.4.1 The protocol

The routing protocol in ActorFrame is actually quite simple. Briefly said there are two main processes involved in the protocol; the periodic transmission of update messages to build the forwarding tables on other connected devices and the maintenance process of periodic cleanups of the forward table.

To realize these two processes different interval timers exist. There is one timer which decides how often the scheduler shall be run, and in ActorFrame this interval timer is 100. This means that every 100 ms there is a check whether it is time to transmit update messages to other routers or whether it is time to update the router. Update messages are transmitted every 100 * 100 ms, which is every 10 seconds (30 seconds in MIDletActorFrame). This message contains routing information gathered from the forward table which is useful for other routers; giving information on actors located on the machine where the router runs, actors on mobile devices connected to this machine and also routing information of all routers known to this router. Routers have their own type of address, namely the RouterAddress. These addresses are *always* included in the update message – the ActorRouterRegMsg. The message is sent to all reachable routers (known routers).

Every 100 seconds the router is updated. This process can also be called the “cleanup”-process. When this timer expires it is time to remove entries in the forward table which is too old. Too old is in this protocol defined to be “since the last cleanup process”, which implies that all entries in the forward table older than 100 seconds is removed. Not having received routing information on a forward table entry in 100 seconds is therefore interpreted like the route no longer exists. An entry is done in the forward table every time a message is received by the router and every time a routing update message is received. Through this and the cleanup process the forward table is built and kept up to date.

4.4.2 Compared to other existing protocols

Comparing the routing protocol to more standard distance vector and link state routing protocols some points are worth mentioning.

4.4.2.1 No route optimization

First of all, there is no route optimization in ActorFrame. This task is left for the IP routing protocol used in the Internet. Messages are forwarded to the destination by sending the message to the Internet Service Provider (ISP) gateway, and from that point ActorFrame has no saying.

In an ActorFrame environment where only wired communication is needed there is no problem with not having route optimization. This is because route optimization is implemented in protocols used in the Internet, either if it is a distance vector protocol like RIP (chapter 4.2.3.1) or a link state protocol like OSPF (chapter 4.2.3.2). The problem is, however, when wireless communication is used – like in our case, when Bluetooth communication is used. If a large MANET is built route optimization might be necessary to reduce end-to-end delay in the network.

4.4.2.2 Periodic updates and cleanup processes

Every 10 seconds (every 30 seconds in MIDletActorFrame) routing information is sent to all known routers. However, not all the information in the forward table is distributed. Only routing information regarding local actors and actors residing on mobile devices connected to the device is distributed to all the known ActorRouters. This way there is limitation on the amount of routing information passed on to other routers in the network from devices running ActorFrame. Routing information about services on a device is therefore only distributed to connected neighbors, which means that devices that are not directly connected do not receive routing information updates on these services. Doing it this way information about how to reach a router is distributed, but information about services offered by a router is only distributed a few hops in the network.

To reach these services despite the fact that the devices are not “connected”, a default gateway is used. This means that if a service (an actor) is not known the messages are

sent to a default gateway. If the default gateway is not directly connected to the target device providing the service the messages will be routed even further to the next default gateway. From this one realize that the default gateways should be chosen wisely to allow messages to reach their target as soon as possible.

MIDletActorFrame based devices distribute the complete forward table to all known Bluetooth, UDP and TCP sessions every 30 seconds.

Periodic updates are most common in distance vector protocols like RIP. In OSPF update messages are sent only when topology changes occur, reducing routing information flow compared to RIP. This kind of algorithm works best in wired communication since frequent topology change is not very common. In MANETs based on Bluetooth routes can go up and down quite frequently, so here the OSPF update algorithm might cause the information flow to increase compared to an algorithm similar to the one found in RIP.

Every 100 seconds a cleanup process is run and routes that are not updated since the last cleanup are deleted. This cleanup process is also found in RIP, although implemented a little different.

4.4.2.3 Good enough for wireless environments?

As have been shown the ActorFrame routing protocol is originally made for wired communication. The question remains; is the protocol suitable for a wireless environment? According to [15] protocols designed for wired environments, with infrequent topology changes and little mobility, are not suitable for ad hoc networking environments. As stated in chapter 4.3.3 these distance vector and link state based routing protocols can not quite catch up with all the link changes, which in turn results in poor convergence and low communication throughput. As the routing protocol in ActorFrame has a lot of similarities with RIP, which is a distance vector protocol, ActorFrames' routing protocol will have the same problems in an ad hoc environment as RIP.

4.4.2.3 Protocol suggestions

In chapter 4.3.3.4 a discussion concerning which kind of wireless routing protocol would best suite ServiceFrame/ActorFrame was done. The discussion outlined that one would have to know more exact what kind of ad hoc network the ServiceFrame based devices would operate in. Considering implementation, two separate routing protocols could run on a ServiceFrame based device; one protocol on the wired interface and a wireless protocol on the wireless interface. This way routing information could be optimally distributed on both the wired and wireless interface. On the wired interface the existing protocol could very well be used. The wireless interface would probably have to implement a variant of either a proactive or reactive routing protocol with some adjustments making the protocol more suitable for a Bluetooth environment.

Using the existing routing protocol will work to some degree, as long as the frequency of the route changes is low, but in a realistic ad hoc network this can not be assumed. In 4.4.2.2 a potential problem with the default gateway solution was mentioned. If all connected devices send routing information to one dedicated default gateway this device would have a complete set of routing information. Now any message will reach its target through this default gateway. If this solution is good depends on network size, since one central default gateway will not scale well.

5 Prototype

The prototype that should be included in this thesis had two main "requirements". First of all it should show how services such as for example playing music stored on mobile phones could be implemented on ServiceFrame based servers acting as Home Gateways. Secondly, the prototype should show how these ServiceFrame based devices could be used to extend the range between the target and source by using the devices as mediators in a multihop ad hoc network.

As discussed earlier in the thesis (chapter 3.1.3.2 and 3.1.5) mobile phones do not support scatternet operation, and hence utilizing the P900 used in this thesis in a multihop ad hoc Bluetooth network was not possible. Since the original plan (figure 22) could not be realized, another approach had to be taken.

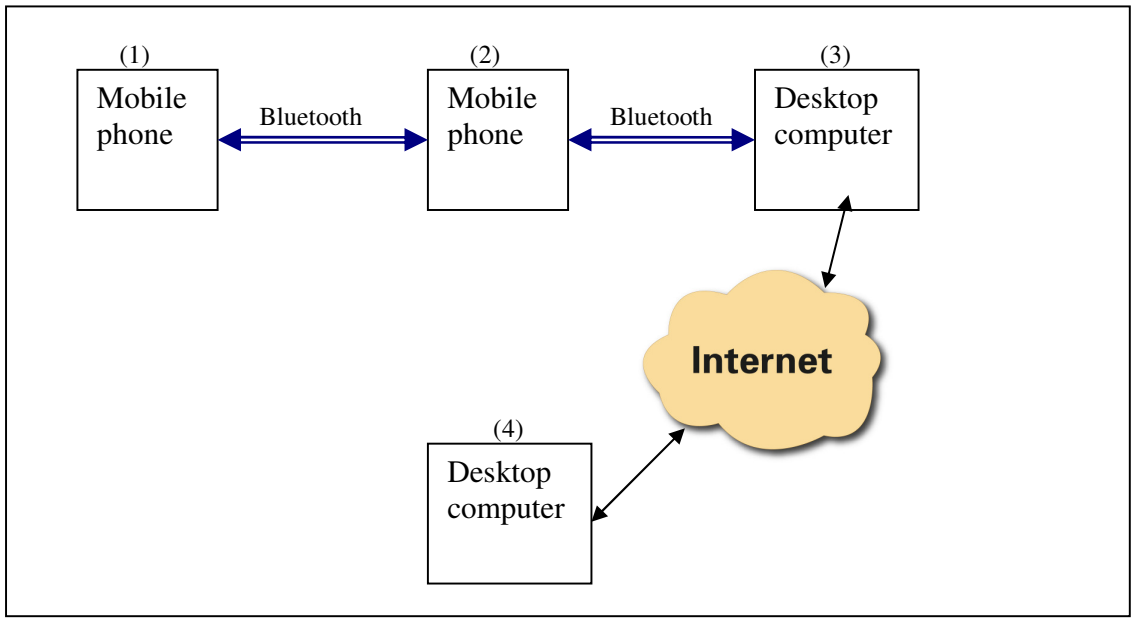


Figure 22: The multihop network that was originally planned to be tested

Device (2) must in this scenario act as a master to device (1) and as a slave to device (3). This is not possible without support for scatternet operation. Therefore an alternative approach, shown in figure 23, had to be used.

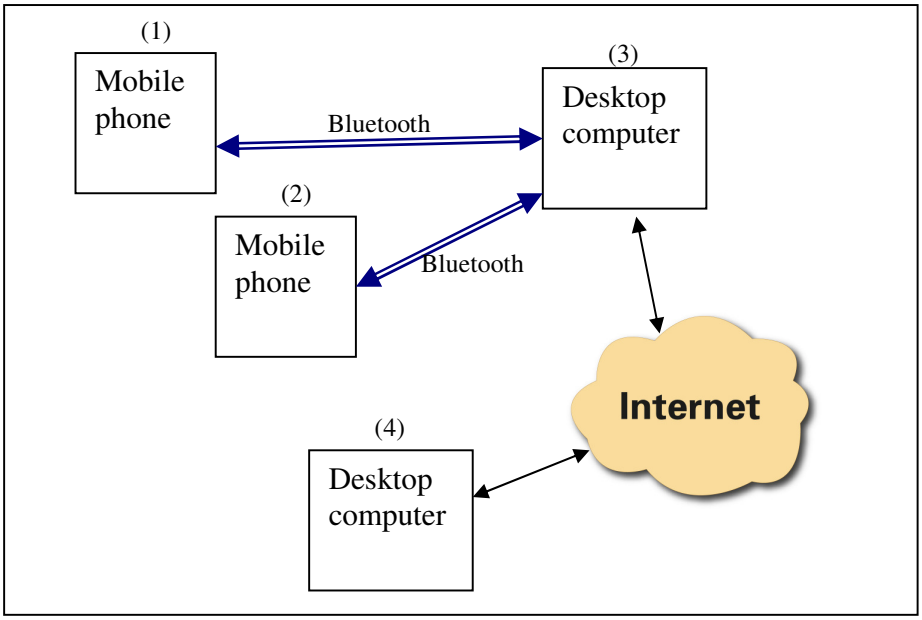


Figure 23: The scenario used in the thesis. Only one master node (device (3)), and two slaves (device (1) and (3)).

The alternative approach is a traditional piconet configuration. This test configuration can, however, still show how a ServiceFrame based device can act as a Home Gateway that can play music stored on a mobile device. From the figure above, this Home Gateway device is device (4).

There are two main solutions to playing music not stored on the local device. One can either download the whole song and play it, or one can stream the song playing it while downloading the file. If one can stream a song, downloading is of course possible. To make a prototype that is a challenge to implement the streaming approach receives most focus. In the following subchapter streaming possibilities are explored.

5.1 Streaming possibilities

There are differences between how streaming possibilities are implemented in J2ME and J2SE, and therefore this chapter will point out some of these.

5.1.1 Streaming in J2ME

Although this thesis should study the possibilities of how a ServiceFrame based *server* can be used as a Home Gateway playing music stored on mobile phones, the possibilities

of doing the playback on mobile devices should be explored. This is because a mobile device actually can act as a ServiceFrame based server, although it would have quite limited processing capability.

It is possible to play back many types of media files on a J2ME based device. However, real-time streaming of media is not easy combined with ActorFrame. In the next two subchapters two ways of realizing real-time media playback in J2ME is presented.

5.1.1.1 Using an Inputstream to create a player

To play media in J2ME one need to use functionality provided by the optional package JSR135, also known as the Mobile Media API (MMAPI). In this API the Manager module is central. Using the Manager one can create a media player by specifying which media file one want to play. Audio data received through streaming is read into an InputStream, and the player can be created:

Player myMP3Player = Manager.createPlayer(InputStream, "audio/mpeg")

Players for other media formats than mp3's are also available. Examples are wave (.wav) audio files, AU (.au) audio files, MIDI files and video files like for example 3gpp and mp4.

One problem with the J2ME implementation of the media player is that it does not support continuous streams, unless the stream is an rtsp stream. This means that the stream the player receives must be fully read before playback can start. To make applications that stream a raw byte stream (not a RTP stream), which the messages in ActorFrame are, to the J2ME device is therefore relatively difficult. One possibility is to have multiple players, where one player starts to play and another player simultaneously loads data to play. This can be illustrated:

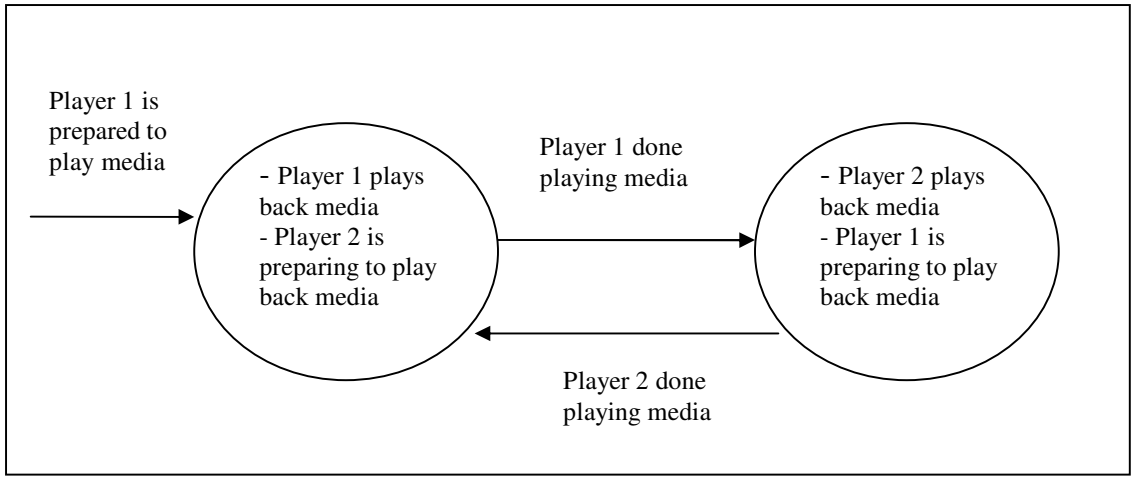


Figure 24: Illustration of a two player solution

The performance of such an implementation, however, is not altogether satisfactory. Every time there is a switch between players there is a glitch, and because of limited processing power on mobile devices this glitch can be substantial.

To explain further why streaming audio, using ActorMsg's, to J2ME using the Player does not work too well, a more detailed outline of the Player implementation is given.

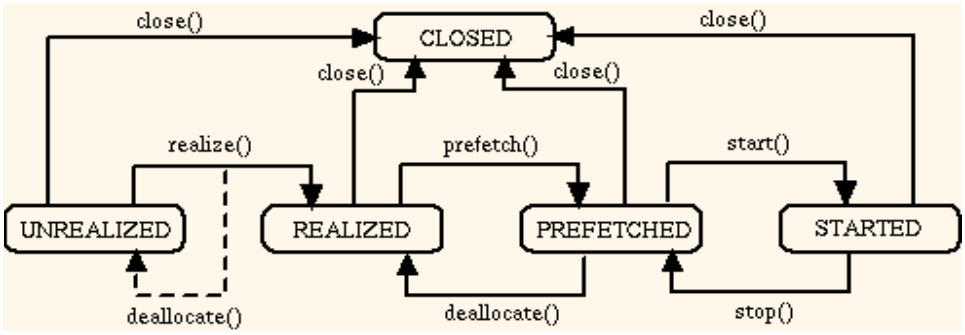


Figure 25: A diagram showing the different states of a Player. [from the API documentation located at http://jcp.org/en/jsr/detail?id=135]

When creating the Player it goes into the UNREALIZED state. An unrealized Player does not have enough information to acquire all the resources it needs to work properly. When it has obtained the information required to acquire the media resources, the Player enters the REALIZED state. The Player may have to communicate with a server, read a file, or interact with a set of objects. This means that the Player reads the Inputstream from the unrealized state to the realized state. Returning to an unrealized state is not possible

unless the realization is not complete. If the realization is complete, which means the Inputstream is fully read, one can never return to an unrealized state. This is the reason why one can not achieve smooth playback without the glitches created by the player switches mentioned above – one simply cannot read the same Inputstream over again to read data received while playing the media.

Before a Player is ready to be started, a Player may still need to perform a number of time-consuming tasks. For example, it may need to acquire scarce or exclusive resources, fill buffers with media data, or perform other start-up processing. This is done by calling the prefetch method on the Player. Calling start() on the Player starts the playback of the media file.

5.1.1.2 Using media locators to create a player

Another option when creating the player is to use media locators. Amongst the media locators supported are file:\\, http:\\ and rtsp:\\. The file protocol is used to play back a media file stored on the local device, and is not relevant for real-time streaming. Using the http protocol is not really real-time streaming since using this protocol actually downloads the entire media file from a web server before it is played. Using the http protocol therefore does not reduce delay from when the user wants to play back the media file to when it actually starts to play. Http is implemented on practically all mobile devices with GPRS/UMTS and J2ME support.

The rtsp protocol is a real-time streaming protocol which actually uses a continuous stream. When using rtsp as the control protocol the most common transport protocol is the Real-Time Protocol (RTP) [37]. This means that playback of the media file starts before the entire media file is downloaded from a server supporting RTP simultaneously as the rest of the file is transferred. This protocol is more and more common on most new mobile devices, enabling them to provide smooth playback of media over a GPRS or UMTS connection.

5.1.1.3 ActorFrame and RTSP

As mentioned in the previous subchapter a possible solution to streaming could be to use the rtsp media locator.

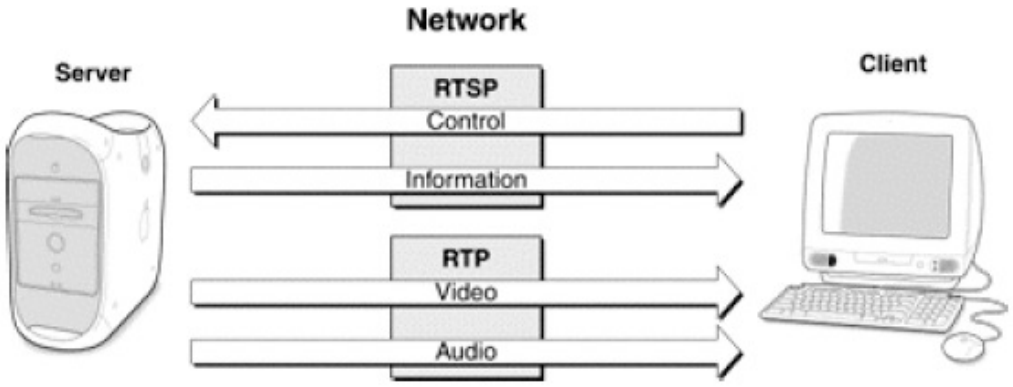


Figure 26: RTP is most commonly used as the transport protocol when using RTSP [37]

However, to be able to realize this solution the ServiceFrame based device must act as an RTP streaming server. Examples of open source streaming servers are Darwin Streaming Server [38] and Helix DNA Server [39]. If one instead of the topology in figure 26 imagines the topology in figure 23 some issues appear. First of all the mobile phone is not likely to run a streaming server like Darwin or Helix, so that streaming can only be made from a desktop computer (acting as a server) to another desktop computer (acting as a client) or mobile phone (acting as a client).

Another problem is the fact that the mobile phone acting as a client is not necessarily directly connected to the streaming server. The reason why this is a problem can be seen on the format of the rtsp locator: *rtsp://host:port/mediafile.* As can be seen use of a locator implies a direct connection. In ActorFrame one must have a dedicated RTP streaming actor, but how to enable the rtsp protocol in Java without using the rtsp locator is an unresolved issue (if even possible), which must be solved to enable ServiceFrame based devices for RTP streaming.

This thesis does not try to solve the issues mentioned above because such a task would be to time consuming considering the other aspects of the thesis. A task like implementing a

streaming server in the ActorFrame framework or integrating existing streaming servers like the ones shown in [38] and [39] could very well be a project assignment in itself.

5.1.2 Streaming in J2SE

Playing media on desktop computers using Java 2 Standard Edition (J2SE) can be done using the Java Media Framework (JMF) elaborated to some detail in 5.1.1.1. In J2SE the same player as presented there is found. There are some differences of course, where the most important one is that the JMF player can not be created using an Inputstream – only media locators (mentioned above in 5.1.1.2). This makes it very hard to realize playback of media using JMF in cooperation with ActorFrame, since ActorFrame messages are serialized and are received as a raw byte stream, and not like a prearranged RTP or HTTP stream.

There is, however, another J2SE API that can be used to play back media received as a raw byte stream, namely the Java Sound API. The playback is done quite similar to the "two player solution" mentioned earlier; where some piece of the media file is played while another piece of it is prepared for playback. Because desktop computers have better processing power than small mobile devices it is suspected that the performance of such an implementation is better than the two player solution for mobile devices. Therefore the glitch which was mentioned when the two player solution was presented in 5.1.1.1 is suspected to be shorter and not as recognizable. This solution and the Java Sound API will be outlined in more detail in chapter 5.2.

5.1.3 J2ME or J2SE implementation?

Since an optimal solution with RTP streaming seemingly is not possible when using ActorFrame, it is not possible to play back an mp3 without some glitches. The best solution would be to have the possibility to both stream to a mobile phone from either a mobile or a desktop computer, and to stream to a desktop computer from a mobile phone or another desktop computer. Therefore this was the intention, but unfortunately playback of mp3 files was not possible with the P900 which was used. The two player solution presented in 5.1.1.1 therefore could not be tested on real devices.

Hence, in this thesis a J2SE solution was designed and implemented. More elaboration on this solution is given in 5.2. Since the solution is not in any way optimal it will be used more as a pointer on ActorFrame performance regarding distribution of routing information and transfer speed. The prototype does of course also provide an indication on whether or not ServiceFrame based devices can play songs stored on other devices, especially mobile devices.

5.2 Design and implementation

The prototype was made quite easy with one actor on the mobile device responding to a streaming request made from a desktop computer, and one actor on a desktop computer requesting and receiving the mp3 file as a stream of ActorMsg's and starting a playback of the media file. First an outline of the actor on the mobile device is done, followed by the description of the design and implementation on the desktop computer. Playback of mp3 files is not originally supported in J2SE. How to enable mp3 support in J2SE is described in Appendix B. Appendix C gives an outline of software and tools used in the implementation.

5.2.1 Mobile phone behavior

The behavior of the actor on the mobile phone, called "Stream", actually only need to react to a request for an mp3 file, divide the file to smaller pieces suitable for streaming and send the file as a stream of messages back to the actor requesting the media file.

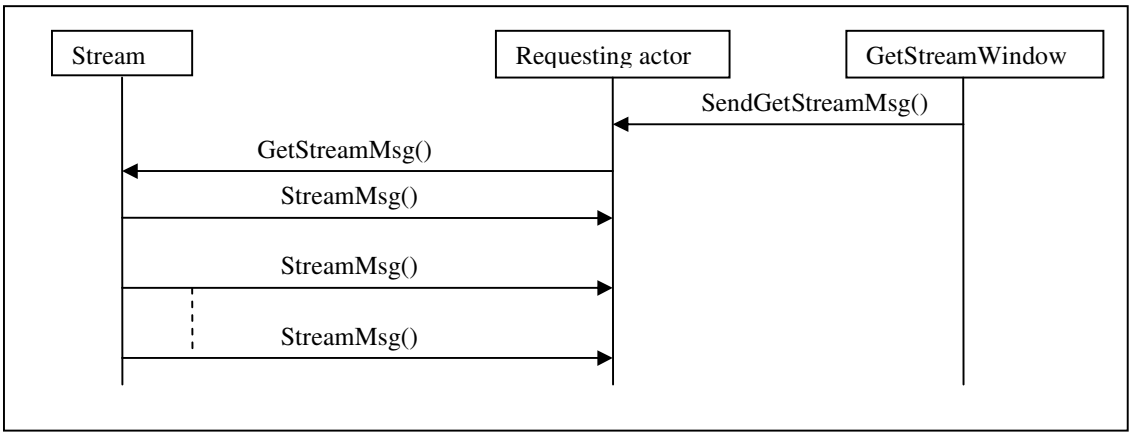


Figure 27: Diagram shows the behavior of the actor on both the mobile phone and the requesting actor (which in this case is a desktop computer which has a user interface called GetStreamWindow). Recalling figure 23 the stream of StreamMsg's containing different parts of the media file is first transmitted using a Bluetooth link, and then further transmitted using either UDP or TCP to the destination.

The division of the file into smaller pieces is done by only reading chunks of data. First the mp3 had to be wrapped into a DataInputStream:

DataInputStream input = getClass().getResourceAsStream("/musicfile.mp3");

After this operation has taken place a series of operations are done to send StreamMsg's containing audio data:

1. Read chunk of data
2. Make StreamMsg
3. Add chunk of data to the StreamMsg
4. Send the StreamMsg

This procedure is repeated until the whole file is read and transmitted, making a stream of StreamMsg's. Appendix G shows code samples of this behavior.

5.2.2 Desktop computer behavior

The behavior of the desktop computer is also quite simple, and is naturally the reverse of the behavior on the mobile phone. This means that the actor, which is called "TestBluetooth", sends a request for an mp3 stream. A simple user interface was made – shown in figure 27 as GetStreamWindow. Three mp3's can be chosen from (with 56, 128 and 320 kbps quality, providing a rather static solution. This is not an important limitation in this case, since the prototype is only supposed to be a test application that shows that playing songs stored on a mobile phone is possible.

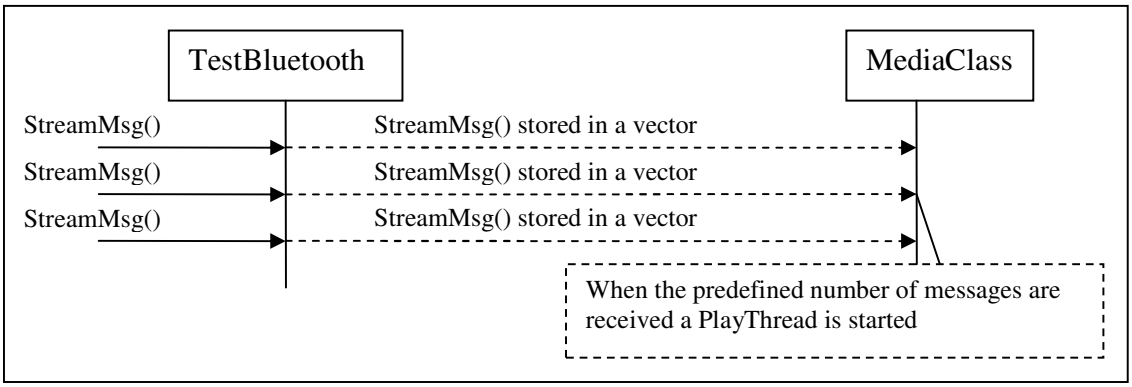


Figure 28: Illustration of how StreamMsg's are handled when received

When this is done a stream of StreamMsg's are received. The task of playing the mp3 file is handled by a class called "MediaClass". Messages are stored in a Vector until a

predefined number of packets are received, making a sort of buffer system. This behavior can be further studied in the code samples provided by Appendix G.

5.2.2.1 Buffering

Having a buffer is useful when the transfer speed is below the playback rate of the mp3 file. According to [40] the de facto standard bitrate on mp3 files is 128 kbps. If one try to stream an mp3 file with this bitrate, but the transfer speed is slower, starting the playback immediately after receiving the first StreamMsg is not a good solution. This would most likely cause the playback to stop relatively fast because the data is "consumed" (played back) faster than it is received, eventually causing audio data shortage. With a vector storing data from a predefined number of StreamMsg's the bitrate of the mp3 can actually exceed the transfer rate for some time without causing the playback to stop.

If the playback should stop because the buffer is too small to even out the difference of the transfer speed and bitrate of the mp3 song, there will be a pause (buffer period) before one once again try to read the buffer. In the prototype a buffer period of 12 seconds is used. No study of optimal buffering time has been done, so this is chosen quite arbitrary. Nevertheless, any longer buffer period might cause the sense of streaming to disappear and a shorter buffer period does not allow a sufficient amount of data to be received.

When the predefined number of packets has been received a Thread handling playback of the media file is started.

5.2.2.2 The PlayThread

Playback of the mp3 file is done making a PlayThread. First of all data from the vector containing the audio data received via the StreamMsg's is wrapped into an inputstream. In the Java Sound API there are some central classes; AudioSystem, AudioInputStream, AudioFormat, DataLine.Info and SourceDataLine (shown in the code samples provided by Appendix G). Together these classes can play back the mp3.

To illustrate how the different classes interact when playback is performed the diagram in figure 29 can be used:

1. From the Inputstream that the audio data in the vector was wrapped in, an AudioInputStream is made using the central AudioSystem module.
2. The format of the audio that just was wrapped into an AudioInputStream must be known.
3. A DataLine.Info object is created using the audio format found in 2.

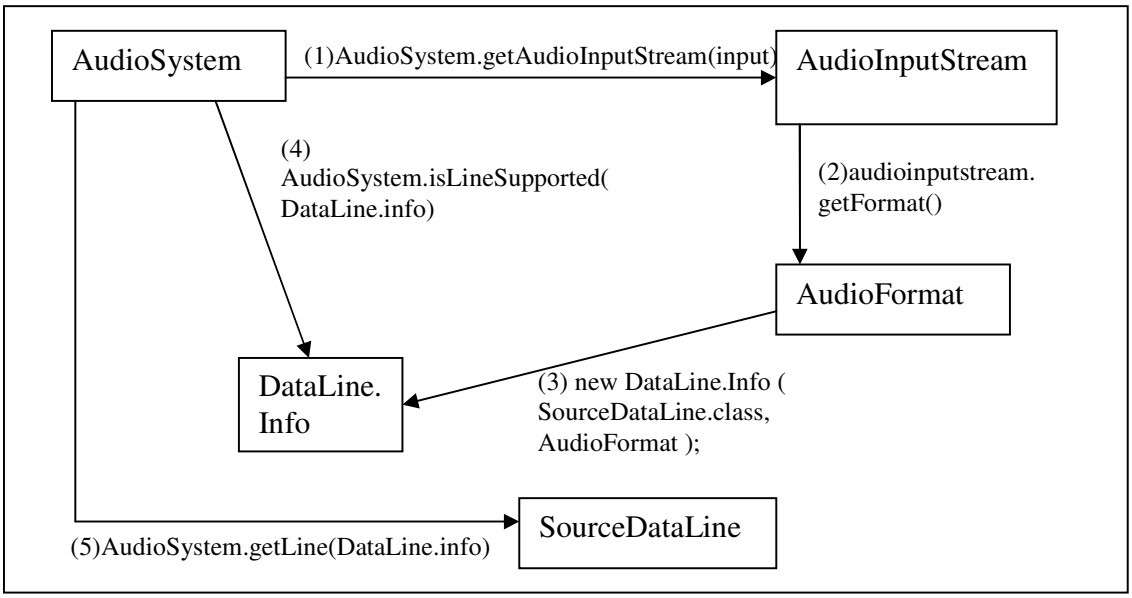


Figure 29: Illustration of PlayThread functionality and the basic classes in Java Sound API

4. The DataLine.Info object created in 3 is used to check whether or not the audio format is supported. If the format is not supported transcoding into a known format can be done, so that playback is possible.
5. Now that it has been made sure that the audio format of the AudioInputStream is known a SourceDataLine object can be created. The mp3 will be played by this object (called "line" below) writing data that is read from the AudioInputStream:

```
while ( ( bytesRead = audioinputstream.read( data, 0, data.length ) ) != -1 ){
        line.write( data, 0, bytesRead );
}
```

The while loop in (5) plays back all the audio data contained in one StreamMsg. When all this data is played back the whole procedure from 1-5 is repeated.

5.2.2.3 Problems with the solution implemented in PlayThread

One soon realizes that the solution presented in 5.2.2.2 is not optimal. Data is not loaded into the AudioInputStream while playing the mp3 and this is a big weakness with the solution. Unfortunately one can not load new data into an inputstream that has already been created. This is the same problem that made it necessary to have two players in the JMF solution in J2ME (chapter 5.1.1.1). The difference here is that there is no player, so that making the same sort of solution is not possible.

Although the solution presented does not provide the optimal playback of an mp3 file, it will work as a pointer on the Bluetooth transfer speed achieved using ActorFrame and it will be able to show that one can use a ServiceFrame based server acting as a Home Gateway to play music stored on a mobile device.

6 Testing

There are several things that should be tested. First of all it should be tested that scatternet operation is not possible as outlined earlier in this thesis (throughout chapter 3). Secondly one should test if the streaming prototype works – to see if a ServiceFrame based desktop computer can play music stored on a mobile phone. Testing whether or not routing information is exchanged properly between devices should also be done. Other things useful to test when evaluating the Bluetooth functionality in ActorFrame is the transfer speed achieved.

6.1 Testing scatternet operation

As [35], [36] and [41] suggests, mobile devices does not support scatternet operation. In addition most mobile devices only support a few active connections at a time. [28] suggests that the newest models from Sony Ericsson only support up to 3 active connections. The P900 actually only support one active connection. When connected to another device the P900 is no longer discoverable for other devices. Some mobile phones from Nokia are supposed to support up to 7 active connections, but this is not confirmed.

Considering these facts there is really no point testing scatternet operation. Multihop ad hoc networking with Bluetooth enabled mobile phones is just not possible. Ad hoc networking is in some degree supported, but only if one can call a piconet an ad hoc network. USB dongles usually have full Bluetooth support with capability of having 7 active connections and where the newest ones actually have support for scatternet operation as well.

With two mobile phones available testing connectivity between a desktop computer acting as a master and two slaves is possible, as shown in figure 31.

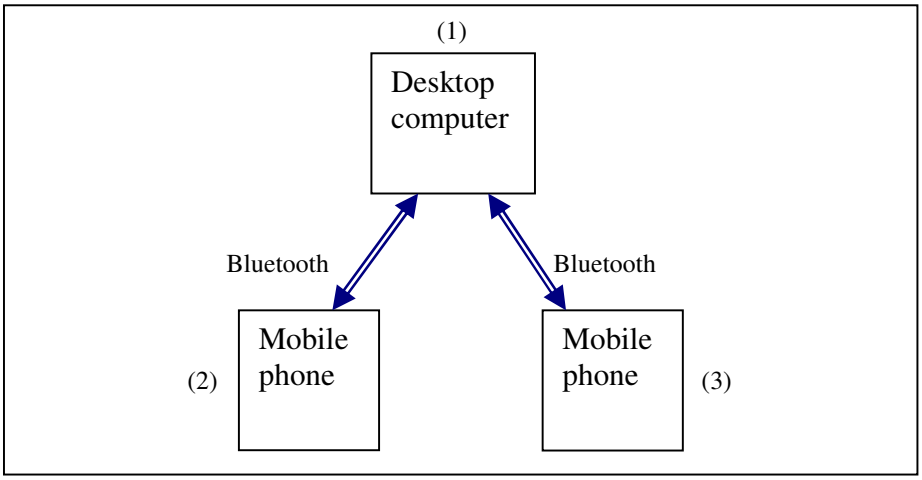


Figure 30: A desktop computer acting as the master with two mobile phones acting as slaves in a piconet

The topology in figure 31 was tested quite a few times, but did not work properly at first. Although a connection was made with both of mobile phones, the listening process implemented could only find and connect to one device. This had to be changed, and this is showed in Appendix D.

It was discovered that the discovery/listening process in both ActorFrame and MIDletActorFrame was ended as soon as a connection was established with a device. Appendix D shows how this had to be changed to allow the discovery/listening process to continue also after a connection is established with a device.

In addition to the connectivity test above connectivity between two mobile phones was tested. This worked, and messages could be exchanged between the mobile phones. None of the phones could now, however, connect two the desktop computer because the P900 only supports one active connection.

6.2 Stream and play song stored on mobile device

A considerable part of this thesis is to see if and how it is possible to play music stored on a mobile device on a ServiceFrame based server.

6.2.1 Topology and configurations

The network topology used to test this is shown in the figure 31. As mentioned in 5.2 the prototype is comprised of the actor called Stream on the mobile device (1) and the actor TestBluetooth on the desktop computer (3). Desktop computer (2) will in this topology only work as a router/mediator.

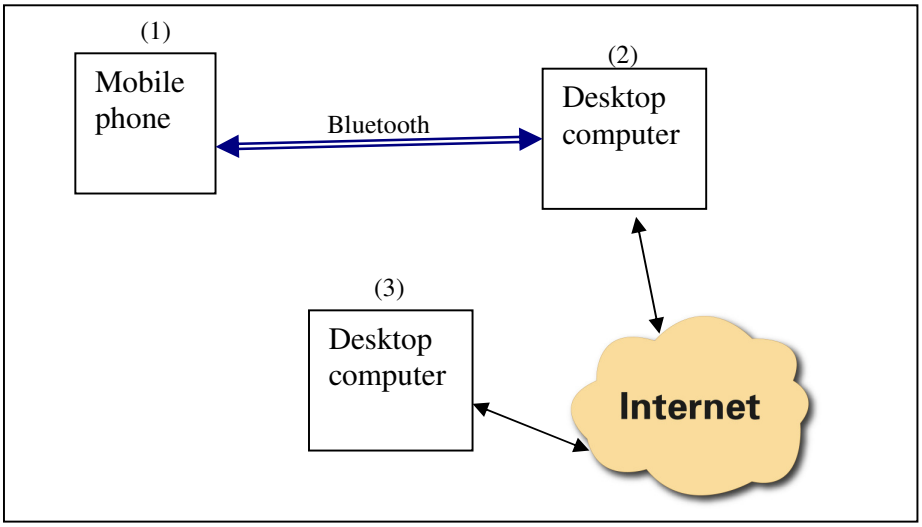


Figure 31: The song is streamed from device (1) to device (3) when testing the prototype

Some property configurations had to be done:

- GATEWAY_IP_ADDRESS (in the property file AFProperties.properties) for device (3) was set to the IP address of device (2). This way device (2) and (3) discover know of each other so that routing information is sent between them. (this was just a choice – could have been done the other way around too)
- Device (2) had to be Bluetooth enabled, which is done by setting the property value of "Bluetooth" to "true" (in the property file AFProperties.properties).
- On device (1) the Bluetooth functionality had to be turned on in MidletContainer.java in the MIDletActorFrame package by setting btSessionEnabled = true

6.2.2 Testing performance

When testing the prototype one expects the mp3 file stored on the mobile phone to be played back on the desktop computer (device (3)). Depending on the quality of the mp3

the streaming will or will not work. In this thesis mp3's with the following bitrates were tested: 56 kbps, 128 kbps and 320 kbps.

The fact that Bluetooth v1.2 supports up to 721 kbps one could expect all these mp3's to be successfully streamed. Some different sizes of the buffers before playback could start were also tested. A buffersize of for example 20 implies that audio data contained in 20 StreamMsg's are buffered. Every StreamMsg contains 58000 bytes of audio data. This means that a buffersize of 20 implies that 58000 * 20 = 1160000 bytes are stored before playback can start.

6.2.2.1 Testing streaming prototype

As can be seen in table 3 playback of the mp3 with a bitrate of 56 kbps was played back successfully. When the bitrate was increased to 128 kbps performance was not too good, and the prototype was next to useless when the bitrate was 320 kbps. This indicates a transfer speed below 128 kbps.

Table 3: Test results from the streaming prototype

Bitrate	Buffersize = 20	Buffersize = 10	Buffersize = 5
56 kbps (file size: 1.65 MB)	Playback is ok	Playback is ok	Playback is ok
128 kbps (file size: 3.0 MB)	Playback is seemingly ok. *	Playback is ok up to a certain point (about 22-26 StreamMsg's) where the buffer is empty. After this buffering is frequent and performance is poor. *	Playback performance is poor. Must frequently stop to buffer. *
320 kbps (file size: 5.56 MB)	Playback performance is poor. Must frequently stop to buffer. *	Same as previous	Same as previous

* The complete file is not received, so an evaluation of this is incomplete (elaborated in 6.2.2.3)

6.2.2.2 Calculate actual transfer speed

Average transfer speed:

In 6.2.3.1 it was discovered that the transfer rate had to be below 128 kbps (because playback stopped to buffer after some StreamMsg's when the mp3 quality was 128 kbps). To discover the average transfer speed from the mobile phone to the desktop computer (device (3) in figure 31), the System.currentTimeMillis() property could be used. Registering the time after the first StreamMsg is received and after the last StreamMsg is received makes it possible to calculate the transfer speed. These calculations were done in MediaClass.java (shown in Appendix F).

The average transfer speed is, as can be seen in figure E-3 in Appendix E, quite low compared to what could be expected from Bluetooth. Comparing the speed to the UMTS technology the speed is rather poor as well.

Transfer speed per StreamMsg:

The transfer speed of each StreamMsg fluctuates, especially in the beginning of a transfer. Figure E-1 in Appendix E show printouts from the beginning of a transfer. Here transfer speed of 80 kbps, 100 kbps and 44.4 kbps is observed. After some time the transfer speed stabilizes around 133 kbps, as can be seen in figure E-2 in Appendix E. The calculations were done in RouterMsg.java (shown in Appendix F)

6.2.2.3 Device and J2ME limitations discovered

A few more limitations with the P900 were discovered during testing of the prototype. One limitation is that to be able to wrap the mp3 into a DataInputStream, as described in 5.2.1, the mp3 had to be split into several files. The reason for this is that the P900 can not make a DataInputStream of a file with size larger than approx 1MB. Therefore a tool called "Easy MP3 Split (trial version is included in Appendix H) had to be used, splitting the mp3. This way the mp3 could be wrapped into a DataInputStream and read into smaller chunks of data, making StreamMsg's containing these chunks of data.

When transmitting the chunks of data from the mobile device a sleep period had to be put in after each chunk was sent, or else the application on the mobile phone would crash.

Also a sleep period had to be put in after a part of the split mp3 was sent (shown in Appendix G). The reason why these sleeps periods had to be put in is unknown, but most likely it is because of memory limitations on the mobile device. If too many StreamMsg's are buffered before transmission the virtual memory capacity is most likely exceeded and the application crashes.

Another limitation with the P900 was revealed. What happened was actually that the application on the mobile phone crashed after transmitting a certain number of StreamMsg's, causing the mp3 to be only partially transmitted. These messages contained a total of approximately 2.5 MB of audio data, with each StreamMsg containing 58000 bytes of audio data. Exactly what caused this error is unknown, but it probably has something to do with limited memory capacity on the mobile device. Therefore the prototype is made so that less than 2.5 MB of audio data is opened. In Appendix A it is shown how this can be changed, if other (newer) mobile devices are available.

6.2.3 Routing information distribution

There is no special way to test if the routing information is distributed. What can be done is to print out actors included in the ActorRouterRegMsg's received (which are the update messages used in ActorFrame). When the topology in 6.2.1 (figure 31) is used the printouts below are available via device (2). In the following the printouts from this figure is outlined. The printouts are shown in Appendix E.

From the mobile phone:

```
Actors: [/stream@Stream, arts@ActorDomain]
```

Information about the actors located on the mobile phone is successfully sent and received.

From device (3):

```
Actors: [129.241.208.90:5557/testbluetooth@TestBluetooth,
129.241.208.221:5557/arts2@ActorRouter,
129.241.208.221:5557/arts@MidletRouter]
```

Information about the actor located on device (3) is successfully sent and received as the first entry shows. The last two entries are included because they are RouterAddresses,

and hence they will always be included in an ActorRouterRegMsg. Entry number two shows that routing information from device (2) has been received, and entry number three shows that routing information from the mobile device has been received.

The printout from the second desktop (device (3)) in the topology shown in figure 31 is shown below.

```
Actors: [129.241.208.90:5557/arts3@ActorRouter, /stream@Stream,
arts@MidletRouter, arts@ActorDomain,
129.241.208.221:5557/arts2@ActorDomain]
```

Entry number 1 and 3 are RouterAddresses and are always included in ActorRouterRegMsg's. Entry number 2 shows that routing information from the mobile phone has reached device (3). The 4th entry shows that routing information on actors locally on device (1) is received. Entry 5 is routing info on actor locally on device (2).

These printouts show that routing information is distributed as outlined in chapter 4.4. One also can see that routing information is not sent from a desktop computer to a mobile phone, since the following entries were missing in the ActorRouterRegMsg received by device (2) from the mobile phone: arts2@ActorRouter, 129.241.208.90:5557/ arts3@ActorRouter. These entries should have been included if routing information from (2) was sent to the mobile phone, because RouterAddresses are always included in an ActorRouterRegMsg. Routing information is not, however, distributed from an ActorRouter (residing on a desktop computer) to a MIDletRouter (residing on a mobile phone) via a Bluetooth link. This is something that should be implemented.

6.3 Discussion of test results

Connectivity worked as suspected. Two mobile phones could connect as slaves to the master device (the desktop computer), and two mobile phones could connect (but then not to the desktop computer). Since this first topology was possible it could be possible to send data to/from the mobile phone to device (3) in figure 32 with device (2) as a mediator. This could only be possible if routing information was properly distributed, and as shown in 6.2 routing information was indeed distributed. The only drawback is that

routing information is not sent from a desktop running ActorFrame to a mobile phone running MIDletActorFrame, but for the prototype this was not necessary.
The prototype did not work very well, mostly because the transfer speed was too low to get a satisfactory playback of the high quality mp3's, but also because the quality of the playback was not optimal. One could have separated the size of the audio data included in the StreamMsg's and the size of the audio data read from the buffering Vector. This way one could have limited the number of reads done from the buffer. In this thesis things were kept very simple, and therefore such a solution was not implemented. Although smooth playback was not accomplished it was shown that it is possible to play a song stored on a mobile device on a desktop computer using ServiceFrame.

7 Discussion and conclusion

In this chapter the accomplished results doing this thesis will be discussed, along with a discussion of the used technology and the prototype.

7.1 Technology, prototype design and performance

In the thesis the implemented routing protocol in ActorFrame was studied, as well as alternative protocols for wireless ad hoc networking. Also through this thesis, it has been discovered shortcomings in both mobile phones implementing J2ME and the Bluetooth standard.

7.1.1 The routing protocols

ActorFrame implements a routing protocol with similarities to more known IP routing protocols. There is however a difference in that the ActorFrame routing protocol does not implement route optimization, a fact that reduces needed processing power. With periodic updates and cleanup processes deleting stale routes the protocol works satisfactory in regular wired communication, in which environment ActorFrame has been used up to now. With the integrated Bluetooth functionality and theoretically possible formations of ad hoc networks this protocol does not satisfy wireless routing protocol specifications. However, as shown through the thesis multihop ad hoc networking is not possible at the present time. This limits the possible topologies to standard piconet solutions, where the need for a special wireless routing protocol is limited; hence the existing ActorFrame protocol actually could work for piconet solutions.

7.1.2 Bluetooth

The Bluetooth specification allows for multiple piconets to connect and form scatternets. This functionality, however, is optional for vendors of Bluetooth chips to implement. Therefore scatternet operation has not yet been included in any Bluetooth chips made for mobile phones, partially because this functionality would demand quite a lot of processing power. Another part of the explanation is that scatternet operation really is not necessary to enable Bluetooth to function as a cable replacement protocol, which is the main goal of Bluetooth [18]. In addition no protocols for neither Bluetooth scatternet

formation nor routing in Bluetooth networks have been standardized (and hence not included in the Bluetooth standard), although many suggestions have been made. Actually most mobile phones do not even support multiple simultaneous Bluetooth connections, which is one of the main requirements for forming scatternets. Putting all these facts together one realizes that forming multihop ad hoc networks is just not possible with today's technology.

7.1.3 Prototype design and performance

The shortcomings in the Bluetooth technology and mobile phone technology made it impossible to test whether or not ServiceFrame based devices works well in a multihop ad hoc network environment. Having that in mind the prototype had to focus on something else. The thesis was to include a study of how music stored on a mobile phone could be played on ServiceFrame based devices and how ServiceFrame based devices could act as mediators. These facts opened the possibility to make a prototype that streams an mp3 from a mobile device to a desktop computer running a ServiceFrame based application.

Studying the streaming possibilities available the prototype design is quite limited. The design does not imply optimal performance since no standard real-time streaming protocol is used. ActorFrame does not integrate any streaming server capability, and hence using the RTSP protocol which is supported in J2ME is not possible through ActorFrame. The design therefore serves more as an example of the fact that music can be streamed from one device and played back on another using ServiceFrame as a framework.

The performance of the prototype was not very good, but as a pointer it works satisfactory. Using the prototype it is possible to test how the performance of the Bluetooth functionality is integrated with the ActorFrame framework. As shown in 6.2.2.2 the average transfer speed was only a little over 100 kbps, which is very bad compared to what could be expected from Bluetooth. The theoretical maximum of Bluetooth v1.2 is 721 kbps. The main reason for the slow transfer speed of only

approximately 1/7 of theoretical maximum is probably that the P900 has quite limited processing power. There is a considerable amount of Threads running in the MIDletActorFrame package; the Bluetooth device and service discovery threads, the Scheduler processes and the router all take up a lot of the processing power that the Bluetooth sending processes needs to work optimally.

If one assumes that Bluetooth v2.0 will experience the same degrading of performance, the transfer speed will be approximately 300 kbps when running ServiceFrame on newer mobile devices with v2.0 support. This is still not enough to stream high quality mp3's, but will stream the "de facto" mp3 quality of 128 kbps without any problems. All in all one can not be satisfied with the performance of the Bluetooth functionality.

Since one of the main objectives of this thesis was to study how ServiceFrame can be used in "a beneficial way to establish ad hoc networks and service sessions", the discovery of the poor performance of the Bluetooth functionality is important. According to Telenor Mobil [43] the highest expected transfer speed in their UMTS network will be 384 kbps. Taking this into consideration there is probably a close race in performance of Bluetooth v2.0 and UMTS, although v2.0 performance is yet to be tested.

UMTS enabled devices can be used wherever you are provided that there is UMTS coverage, as opposed to Bluetooth that has a very short range and needs other Bluetooth enabled devices in the near vicinity. One can therefore most likely limit the use of Bluetooth to indoor use connecting a desktop computer to one or more mobile devices, using the desktop computer almost as an access point. Without scatternet support one can not really use Bluetooth to much more than this. UMTS does indeed cost money to use, but is much more flexible. There is no range limitation giving the possibility to have access to media streaming wherever you are. Streaming services for UMTS already exists and work quite well. These things considered it is possible that the Bluetooth performance is just too poor to use for streaming purposes. For downloading services however, where real-time performance is not that important, Bluetooth functionality could be beneficial compared to UMTS.

7.2 Future work

First of all testing the Bluetooth functionality provided by ServiceFrame with Bluetooth v2.0 compatible devices is of a great importance. To really be able to compare Bluetooth performance with UTMS performance this is necessary.

The prototype can be extended. Now there is a possibility for choosing which mp3 to stream, but it must be done from a predefined list of mp3's; which is a very static solution. A better solution is for example first to request a list of available mp3's. From this list one can choose which mp3 to stream. This would be a much more user-friendly solution.

To enable multihop ad hoc networks using Bluetooth enabled mobile devices scatternet operation must be implemented by the Bluetooth chip vendors. One can hope that this functionality will be implemented as soon as possible, but first a larger part of the consumers must demand scatternet operation functionality.

A more thorough study of the environments in which ServiceFrame based devices would operate could be useful. This way the choice and design Bluetooth scatternet formation protocols and routing protocols would be made on a more substantial ground.

There exist several open source streaming servers (Darwin [38], Helix [39]) which implements standardized real-time streaming protocols – enabling smooth playback of media. A study of how servers like these could be combined with ActorFrame to enable streaming server capabilities to an ActorRouter would be interesting. If an ActorRouter could act as a streaming server new streaming services could be developed.

7.3 Conclusion

Through the work on this thesis it has been discovered that there exists a lot of limitations in the implemented functionality on mobile devices, first of all regarding Bluetooth functionality. Before multihop ad hoc networking is possible with mobile devices using Bluetooth as the communication protocol there is some functionality that has to be

implemented by the vendors. Among this functionality is for example support for multiple simultaneous Bluetooth connections, support for master/slave switching and support for scatternet operation.

Bluetooth scatternet formation protocols were also studied in the thesis. When support for the functionality mentioned above is implemented a choice regarding which formation protocol to implement must be made. The same goes for which routing protocol to implement on the wireless interface of ServiceFrame based devices. For now the routing protocol implemented can be sufficient as the network topology is limited to piconets, which implies a limited amount of routing information.

With a more thorough study of streaming and streaming servers more optimal streaming services can surely be made using the ServiceFrame framework. This thesis showed successfully that songs stored on a mobile phone could be streamed and played back on a desktop computer running a ServiceFrame based applications. The drawback with ServiceFrame is the processing power required to run applications based on the framework, causing the transfer speed of Bluetooth to be unreasonably low.

With more time the mobile devices will surely be more powerful and support the required functionality, and when this time comes ServiceFrame and Bluetooth will be a good combination for streaming services. Right now, however, the mobile devices have limited processing power and the support for Bluetooth functionality is also limited. This causes the applications made for this combination to be limited by the boundaries given by these technologies.

References

[1] Teleservice lab, Item, NTNU, (URL: http://www.item.ntnu.no/lab/nettint1/index.html)

[2] G. Melby, "ActorFrame Developers Guide", NorARC, ARTS, august 2004 (included in Appendix H as "References/ActorFrame Developers Guide.doc")

[3] Sun Microsystems,"Java APIs for Bluetooth – Wireless Technology (JSR-82)", version 1.1, 2005 (included in Appendix H as "References/JSR82spec.pdf")

[4] Sony Ericsson, P900 Whitepaper (URL: http://developer.sonyericsson.com/getDocument.do?docId=65064, and included in Appendix H as "References/p900Whitepaper.pdf")

[5] Ø.Haugen, "JavaFrame Modelling Guidelines 2.5", Ericsson NorARC, 2001 (included in Appendix H as "References/JFGuidelines.pdf")

[6] Ø.Haugen, B. Møller-Pedersen, "JavaFrame: Framework for Java Enabled Modelling", (URL: http://www.item.ntnu.no/fag/ttm4160/Implementation/ECSE2000JavaFrame.pdf, and included in Appendix H as "References/javaframe.pdf")

[7] G. Melby, Presentasjon av ActorFrame for ITEM-studenter – "ActorFrame – Et Java basert miljø fro modulære nettverksapplikasjoner", NorARC, 2004 (URL: http://www.item.ntnu.no/lab/nettint1/activities/prosjekter/host2004/overview.html and included in Appendix H as "References/modular.pdf")

[8] K.E. Husa, R.Bræk, G. Melby,"ServiceFrame Whitepaper", Ericsson NorARC, 2002 (included in Appendix H as "References/ServiceFrameWhitepaperv8.pdf")

[9] K.E. Husa, R.Bræk, G. Melby,"ServiceFrame and ActorFrame", Ericsson NorARC, 2002 (URL: http://www.item.ntnu.no/fag/ttm4160/ServicePlatforms/ServiceFrame2002.pdf and included in Appendix H as "References/serviceframe2002.pdf")

[10] A. Herstad, G. Melby, "ServiceFrame", 2004 (included in Appendix H as "References/ServiceFrame.doc")

[11] S. S. Kristiansen, "Transparent Communication over Bluetooth", Project assignment, 2005 (included in Appendix H as "References/rapport v1.0.pdf")

[12] Wireless Application Programming with J2ME and Bluetooth (URL: http://developers.sun.com/techtopics/mobility/midp/articles/bluetooth1 and included in Appendix H as "References/j2meProg.pdf")

[13] B. Hopkins & R. Antony, "Bluetooth for Java", Apress, 2003

[14] J. Knudsen, "Wireless Java: Developing with J2ME", Apress, 2003

[15] C.K Toh, "Ad Hoc Mobile Wireless Networks", Prentice Hall, 2002

[16] R. Morrow, "Bluetooth Operation and Use", McGraw Hill, 2002

[17] J. Schiller, "Mobile Communications", Addison-Wesley, 2nd edition, 2003

[18] Bluetooth Special Interest Group, "Specification of the Bluetooth system", v2.0 + EDR, 4.november 2004 (included in Appendix H as "References/ Core v2.0 + EDR")

[19] I. Stojmenovic, N.Zaguia, ”Bluetooth scatternet formation in ad hoc wireless networks”, Chapter 9 in: Performance Modeling and Analysis of Bluetooth Networks: Network Formation, Polling, Scheduling, and Traffic Control (J. Misic and V. Misic), Auerbach Publications (Taylor & Francis Group), 2006, 147-171 (included in Appendix H as “References/BSF-survey.pdf”)

[20] T.Y Lin, Y.C Tseng, K.M Chang, ”A new BlueRing scatternet topology for Bluetooth with its formation , routing and maintenance protocols”, John Wiley & Sons Ltd, 2003 (included in Appendix H as “References/bluering.pdf”)

[21] G.V Zàruba, S. Basagni, I. Chlamtac, ”Bluetrees – Scatternet formation to enable Bluetooth-based Ad hoc networks”, University of Dallas, Texas, Center for Advanced Telecommunications Systems and Services (CATSS), Proc. IEEE ICC, 2001, pp.273-277 (included in Appendix H as “References/blutrees.pdf”)

[22] Z.Wang, R.J Thomas, Z. Haas, ”Bluenet, a new scatternet formation scheme”, Proc. Hawaii Int’l Conf System Sciences, 2002, pp. 721-736 (included in Appendix H as “References/bluenet.pdf”)

[23] S.G Valenzuela, S.T Vuong, C.C.M Leung, ”Mobile BlueScouts: A scatternet formation protocol based on mobile agents”, 4th Workshop on Applications and Services in Wireless Networks, Boston, Aug. 2004 (included in Appendix H as “References/bluescout.pdf”)

[24] Self configuring systems, lecture from students, (URL: http://www.item.ntnu.no/fag/ttm47ac/topics/Bluetooth.ppt and included in Appendix H as “References/Bluetooth.ppt”)

[25] Sun Microsystems, “The Java platform for consumer and embedded devices” (URL: http://java.sun.com/j2me/docs/j2me-ds.pdf and included in Appendix H as “References/j2me-ds.pdf”)

[26] Tim Lindholm & Frank Yellin, The Java Virtual Machine Specifications, Second edition (URL: http://java.sun.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html)

[27] J2ME tutorial, "Part 1, Creating MIDlets" (url:http://today.java.net/pub/a/today/2005/02/09/j2me1.html?page=last and included in Appendix H as "References/j2meTut.pdf")

[28] Link to forum for developers using mobile phones from Sony Ericsson: http://developer.sonyericsson.com/thread.jspa?threadID=18866&tstart=0'

[29] M. P. Clark, "Data Networks, IP and the Internet", Wiley, 2003

[30] U. Black, "IP Routing Protocols", Prentice Hall, 2000

[31] K. Moldeklev, lecture notes TTM4150, 31.08.2004

[32] J. Moy, "OSPF protocol analysis", July 1991

[33] S. Basagni, M.Conti, S. Giordano, I. Stojmenovic, "Mobile Ad-hoc networking", IEEE Press, 2004

[34] Ericsson, "Baseband vs. Host Stack Implementation, Scatternet Part1", June 2004 (included in Appendix H as "References/ ericsson_scatternet_whitepaper_040624.pdf")

[35] L.Strand, A.Tønnesen, "Blåtann/802.15 (WPAN)", 2004 (included in Appendix H as "References/scatternetErTeori.pdf")

[36] Sony Ericsson Developer Forum, 2006 (http://developer.sonyericsson.com/thread.jspa?messageID=76945&#76945)

[37] C.Perkins, "RTP – Audio and video for the Internet", Addison-Wesley, 2003

[38] Apple Developer Connection, ”QuickTime Streaming Guide”, 2006 (included in Appendix H as “References/StreamingGuide.pdf”)

[39] Helix DNA Server Architecture, 2006 (URL: https://helix-server.helixcommunity.org/2003/devdocs/architecture.html)

[40] Wikipedia - the free encyclopedia, 2006 (http://en.wikipedia.org/wiki/Mp3)

[41] F.Kargl, S.Ribhegge, S.Schlott, M.Weber, ”Bluetooth-based Ad-hoc networks for voice transmission”, Proceedings of the 36th Hawaii International Conference on System Sciences, 2003 (included in Appendix H as “References/BTAdHoc.pdf”)

[42] Java Community Process, ”JSR 259 Ad Hoc Networking API”, Draft 0.1.0, 2006 (included in Appendix H as “References/JSR259.pdf”)

[43] Telenor Mobil, UMTS performance, (URL: http://telenormobil.no/tjenester/3g/merom.do) and Wikipedia: (URL: http://en.wikipedia.org/wiki/Umts)

Appendix A: User manual

A-1 Software needed:

- Eclipse IDE, can be downloaded from http://www.eclipse.org/downloads/index.php
- Java Bluetooth API, can be downloaded from http://sourceforge.net/projects/bluecove/, but is included in Appendix H in commonLib/ext

A-2 Starting the JVM version of ActorFrame on a PC

Below a list of things that must be done is order to get the prototype running. This list is valid for both starting the part of the prototype just handling routing and also the part playing the streamed mp3. Enabling MP3 support is outlined in Appendix B and this must be done to get the streaming of the mp3 to work.

- Follow the instructions in Appendix A in [11] to install the Java Bluetooth API
- Open Eclipse and make a new workspace, for example "ActorFrame".
- Right click on white field in the "Package explorer" and choose "Import..". Import the file called "EclipseRouting.zip" to install the part that handles routing or import "EclipseStreaming.zip" (essentially the same packages, except that EclipseStreaming is configured to act as the device receiving the StreamMsg's and has enabled the graphical user interface used to send requests for an MP3) to install the part handling playback from Appendix H and click next. Your screen should then look something like this:

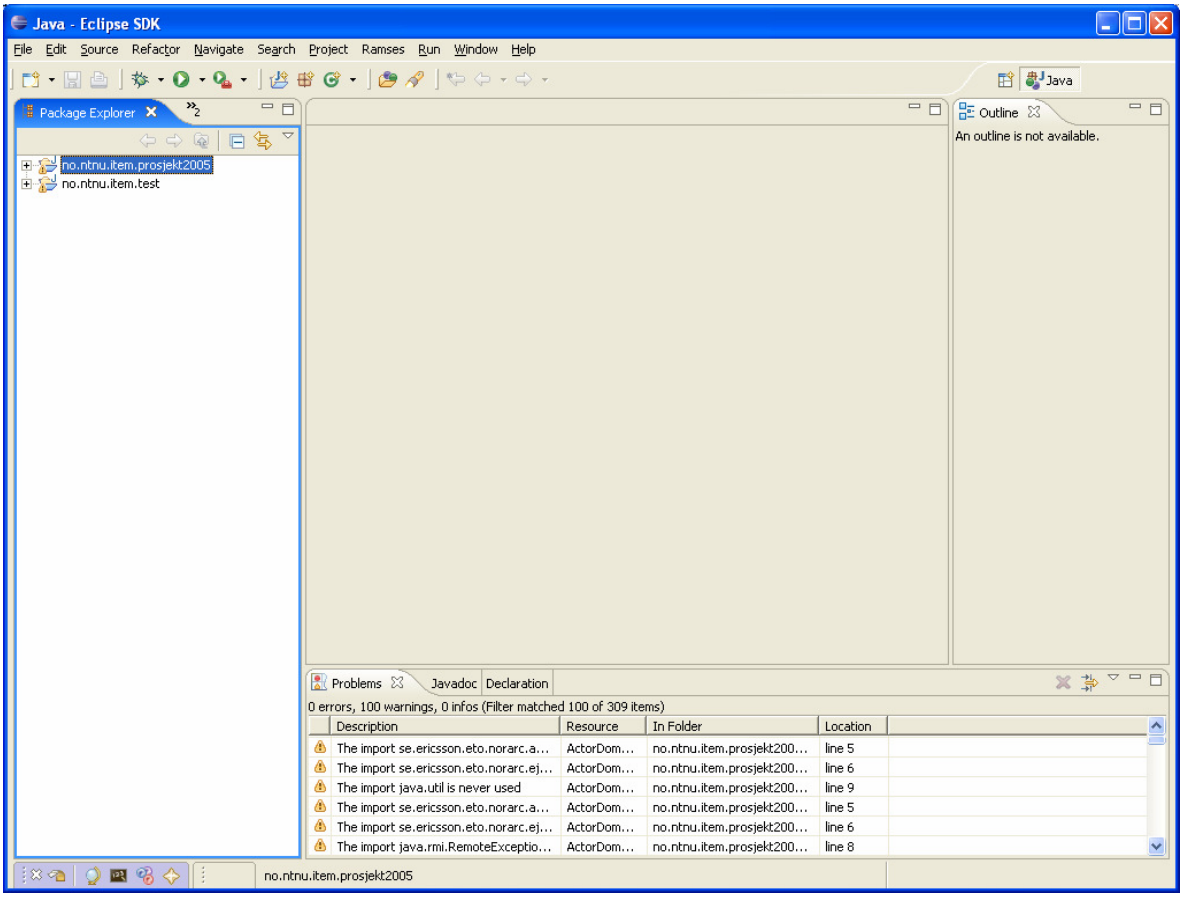


Figure A-1: Screenshot that shows how the screen should look like

If there are build path errors:

- Right click on the project called no.ntnu.item.prosjekt2005, click on properties, and choose “Java Build Path”. The paths to the Libraries need to be edited. Choose the correct path to each of the libraries (“Edit..”): j2ee.jar, jdom.jar, log4j-1.2.12.jar, org.mortbay.jetty.jar and BlueCoveJSR82-Patched-By-Benhui-net.jar, included in Appendix H (commonLib).
- Right click on the project called no.ntnu.item.test, click on properties, and choose “Java Build Path”. The paths to the j2ee.jar library need to be edited. Choose the correct path to /commonLib/ext/j2ee.jar (from commonLib included in Appendix H)

To run the prototype - find the folder called se.ericsson.eto.norarc.standalone in the project called no.ntnu.item.prosjekt2005. Right click on StandAlone.java, choose

"Run as" and click on "Run...". Double-click on "Java-application", which appears as an alternative under "Configurations". Then this is what you should see:

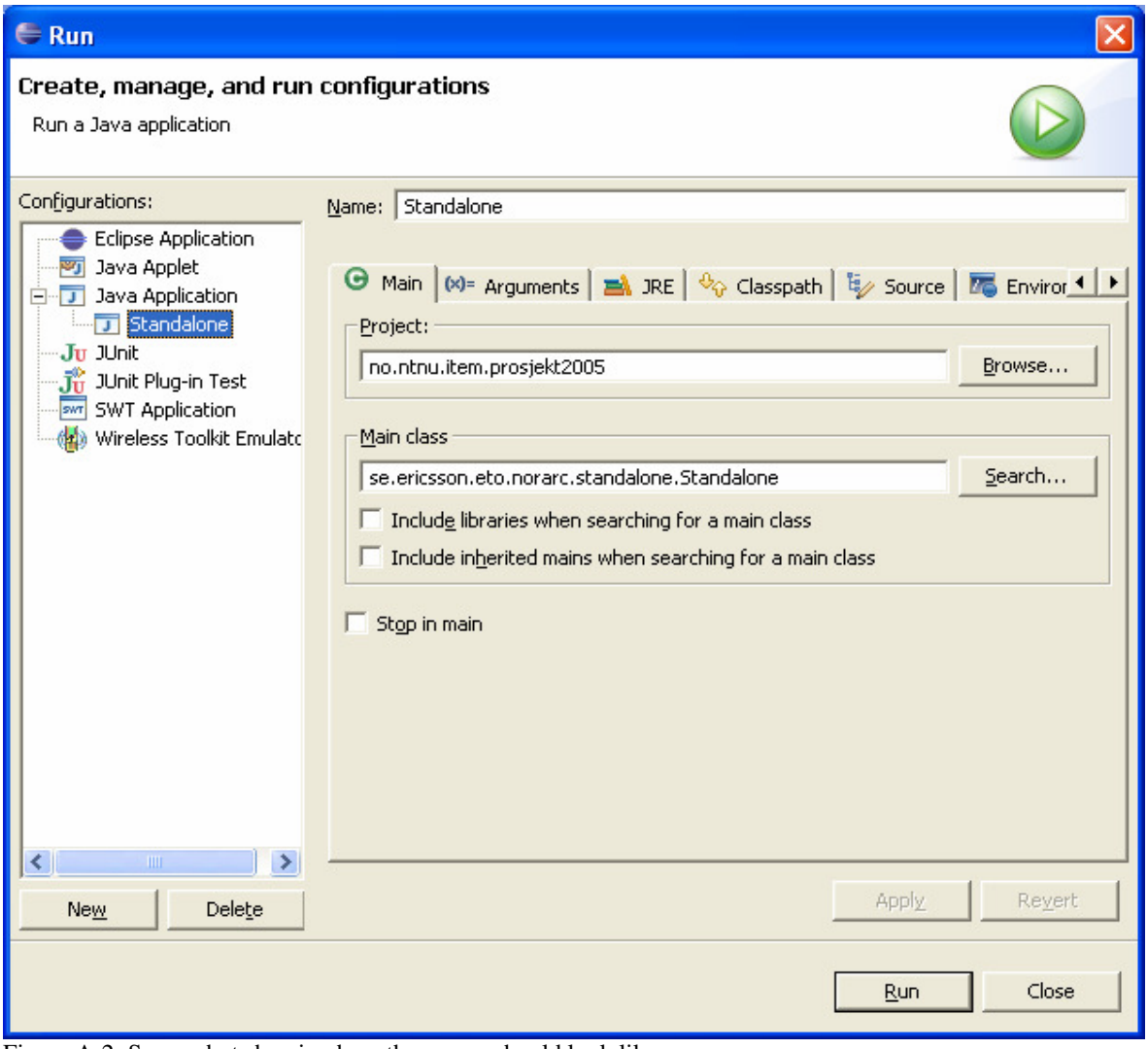


Figure A-2: Screenshot showing how the screen should look like

Now you must click on "Browse…" under "Project". Choose no.ntnu.item.test, click "OK" and click "Run". The prototype should now start. Remember that the mobile device and the router should be connected before the streaming module is started. This way one ensures that all routing information is available for the streaming module, so that streaming can start 60 seconds after the module is started.

To enable exchanging of routing information the GATEWAY_IP_ADDRESS in AFProperties.properties included must be changed. The desktop computer which installs the "EclipseStreaming.zip" package must set this field to the IP address of the desktop

computer that installed the “EclipseRouting.zip” package. To do this open AFProperties.properties and change the GATEWAY_IP_ADDRESS field:

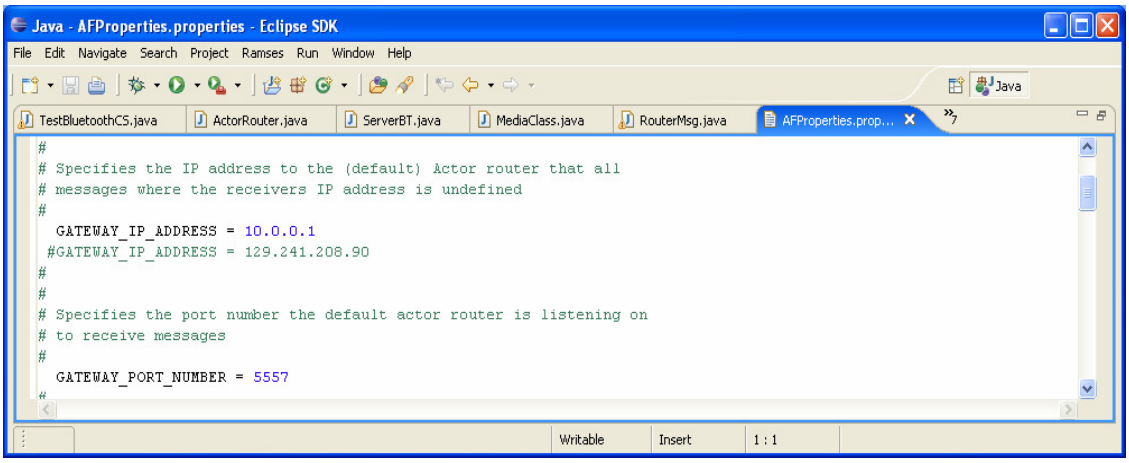


Figure A-3: Screenshot showing which field to change in AFProperties.properties

A-3 Starting the MIDletActorFrame version on the mobile phone:

This does not necessarily require as much configuration, but it requires that you can transfer a JAR file to the mobile phone and install the MIDlet.

Procedure:

- Extract the zip file called “StreamMIDlet.zip” in Appendix H on your computer. Transfer “NyMAF.jar” from <<your path>>/ NyMAF /bin/ NyMAF.jar to the mobile, and install it.
- Start the installed MIDlet (StreamMIDlet)

If the router part of the prototype is running the desktop computer running the router and the mobile device running the installed MIDlet should connect. When connected the streaming module of the prototype can request an mp3 from the mobile phone.

If one wants to test if the mobile device at hand can handle more audio data than the P900 one can try to change the code in StreamCS.java in <<your path>>/ NyMAF/src/actor/stream/StreamCS.java. Here instructions are included as to what variables can be changed to test if the device can handle more audio data.

To run this new edited version of the code a new MIDlet has to be made. In this thesis Wireless Toolkit was used, and in Appendix C an example on how to make a MIDlet is shown.

Appendix B: Enabling MP3 support in J2SE

To enable MP3 support three libraries must be available: jl1.0.jar (Java Layer 1.0), mp3spi1.9.4.jar and tritonus_share.jar (all included in Appendix H). These are all libraries made to enable playback of mp3 files in J2SE, and they are all freeware.

Enabling mp3 support is done by including the jar files in the CLASSPATH. In Windows XP this is done by the following steps: "Control Panel" → "Performance and maintenance"→"System" → the tab called "Advanced" → "Environmental variables. This screen looks like this:

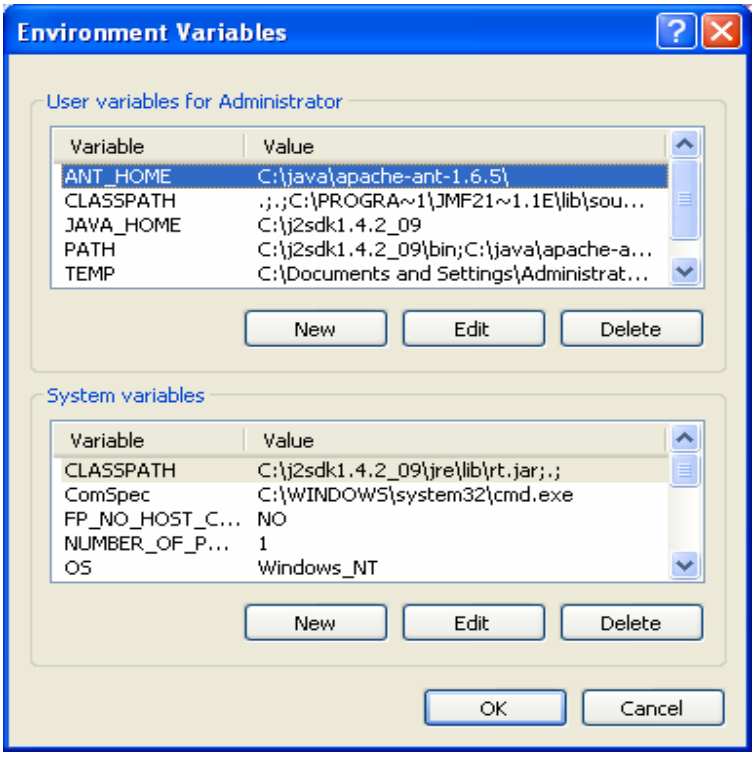


Figure B-1: Changing the environment variables

Now mark the CLASSPATH field and press the “Edit” button:

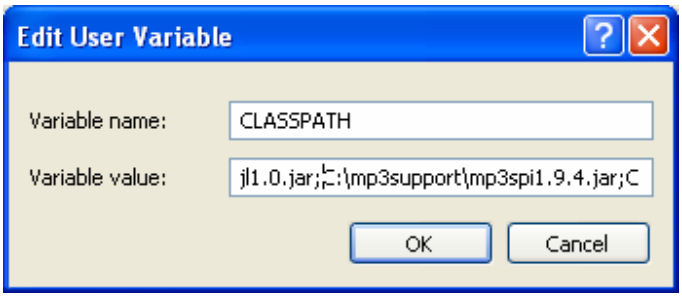


Figure B-2: Changing the classpath

Add the following to the classpath: <<your path>>/<<filename.jar>>, where filename is either jl1.0, mp3spi1.9.4 or tritonus_share. <<Your path>> is where the .jar files are stored on the computer.

Appendix C: Software and tools

Mainly two tools were used: Eclipse IDE and Sony Ericsson Wireless Toolkit. In addition a text editor was used to write the code for the mobile application, which was compiled and packed into a MIDlet using Wireless Toolkit. Below an outline on how to make a .jar file from the code included in Appendix H is given.

C-1 Make a JAR file from java code

Follow the steps:

1. Install the Sony Ericsson Wireless Toolkit (download from: http://developer.sonyericsson.com/site/global/docstools/java/p_java.jsp)
2. Open the KToolbar by following these steps (Windows XP): "Start"→"All programs"→"Sony Ericsson"→"J2ME SDK"→"WTK 2"→"KToolbar".
3. Choose "New project…". Set project name to whatever you want, and set MIDlet class name to "application.Stream.StreamMIDlet". Create project.
4. Extract "StreamMIDlet.zip" from Appendix H. Copy the src from <<extracted path>>/NyMAF/ to <<path where Sony Ericssson WTK is installed>>/ SonyEricsson/J2ME_SDK/PC_Emulation/WTK2/apps/NyMAF. Now the project created in 3 has java code. Do changes in code mentioned in A-3.
5. In KToolbar, choose "Project"→"Package"→"Create package"
6. Now the .jar file is made and is located here: <<path where Sony Ericssson WTK is installed>>/ SonyEricsson/J2ME_SDK/PC_Emulation/WTK2/apps/NyMAF/bin/NyMAF.jar

Following these steps the code can be changed to for example test if the mobile device at hand can handle more audio data than the P900, as explained in Appendix A-3.

Appendix D: Changes in BluetoothListener.java

Some changes were made in BluetoothListener.java in both ActorFrame and MIDletActorFrame. This was done to make sure that device discovery/service discovery continued after having connected to one device. A complete outline is not given here – only a brief look at the changes.

When the work on this thesis started the search for devices in BluetoothListener.java did not continue when a connection to a device was established. This had to be changed so that although a service was found and a connection was established a device discovery process must be started. By comparing the serviceSearchCompleted method of BluetoothListener.java this change is illustrated.

Before:

```
public void serviceSearchCompleted(int transId, int complete){
                        log("service discovery completed with return code:"+complete);
                        log(""+services.size()+" services are discovered");
                                                serviceReturnCode = complete;
                        if(!serviceDiscoveryOK){
                                btlistener.wakeup();
                         }
                         else{
                                searchIndex = 0;
                         }
                }
```

As can be seen here the device discovery process stops if services have been discovered. This and some other things were changed, as shown in figure D-1. For example if all the devices have been searched for services and no service has been discovered a new device discovery is started. If a service has been found (serviceDiscoveryOK=true) and all devices has been searched for services a new device discovery starts – this makes sure device/service discovery is continuously running on the device.

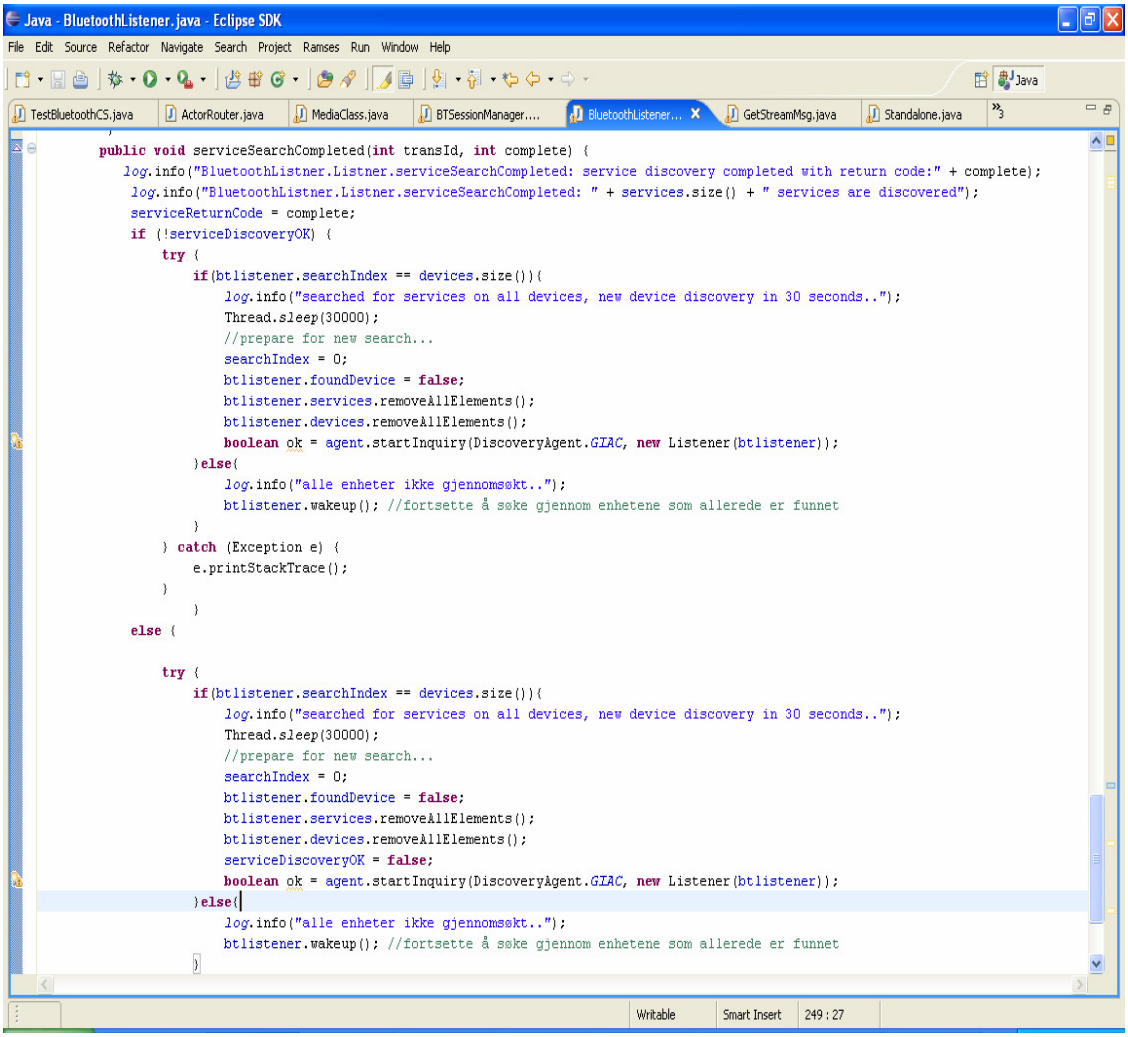


Figure D-1: New code for BluetoothListener.java

Appendix E: Printouts from testing

Below figures belonging to chapter 6.2.2 is given:

The beginning of a streaming session:

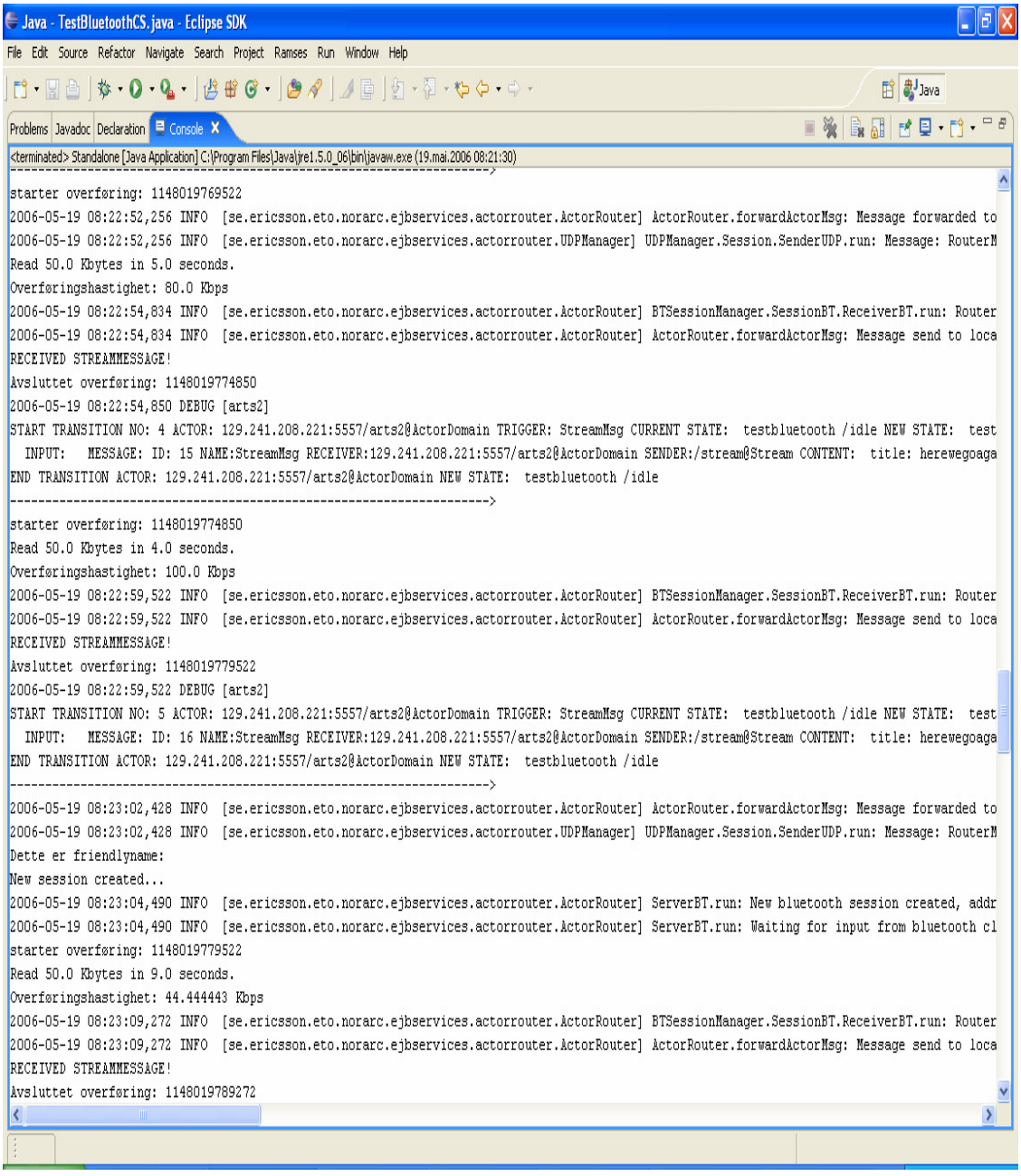


Figure E-1: Transfer speed per StreamMsg fluctuates

Transfer speed stabilizes after some StreamMsg's:

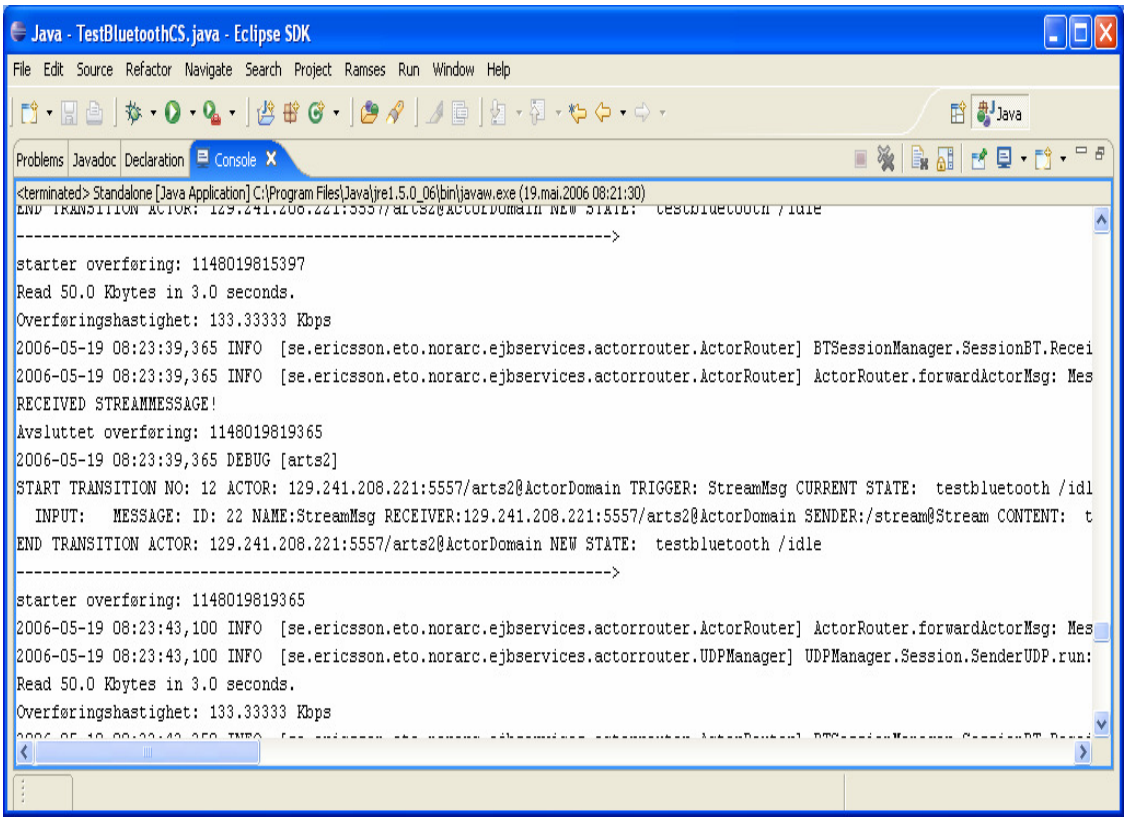


Figure E-2: Transfer speed stabilizes after some StreamMsg's have been received

Average transfer speed:

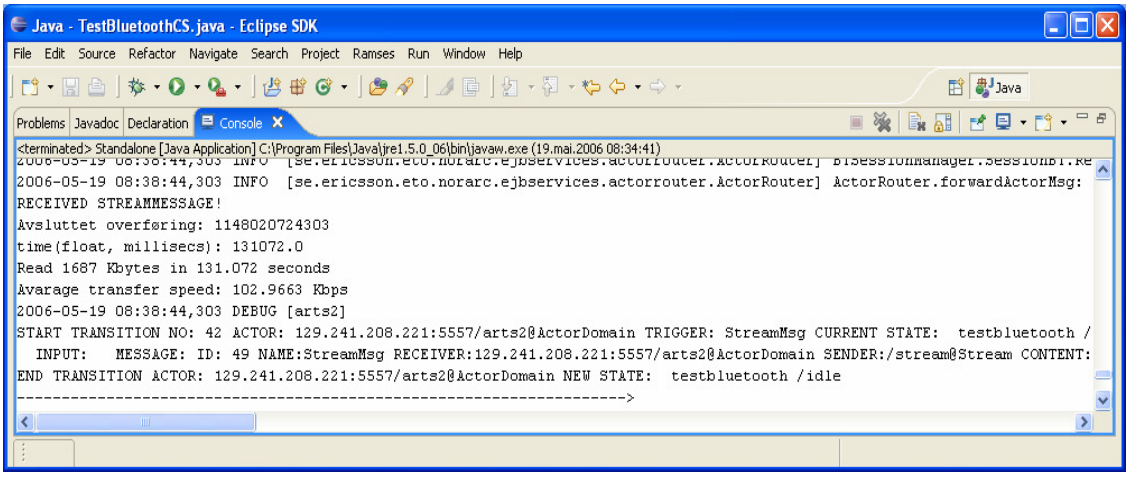


Figure E-3: The average transfer speed is calculated to approximately 103 kbps

Printouts from the desktop computer (device(2) in figure 31, chapter 6.2.1):

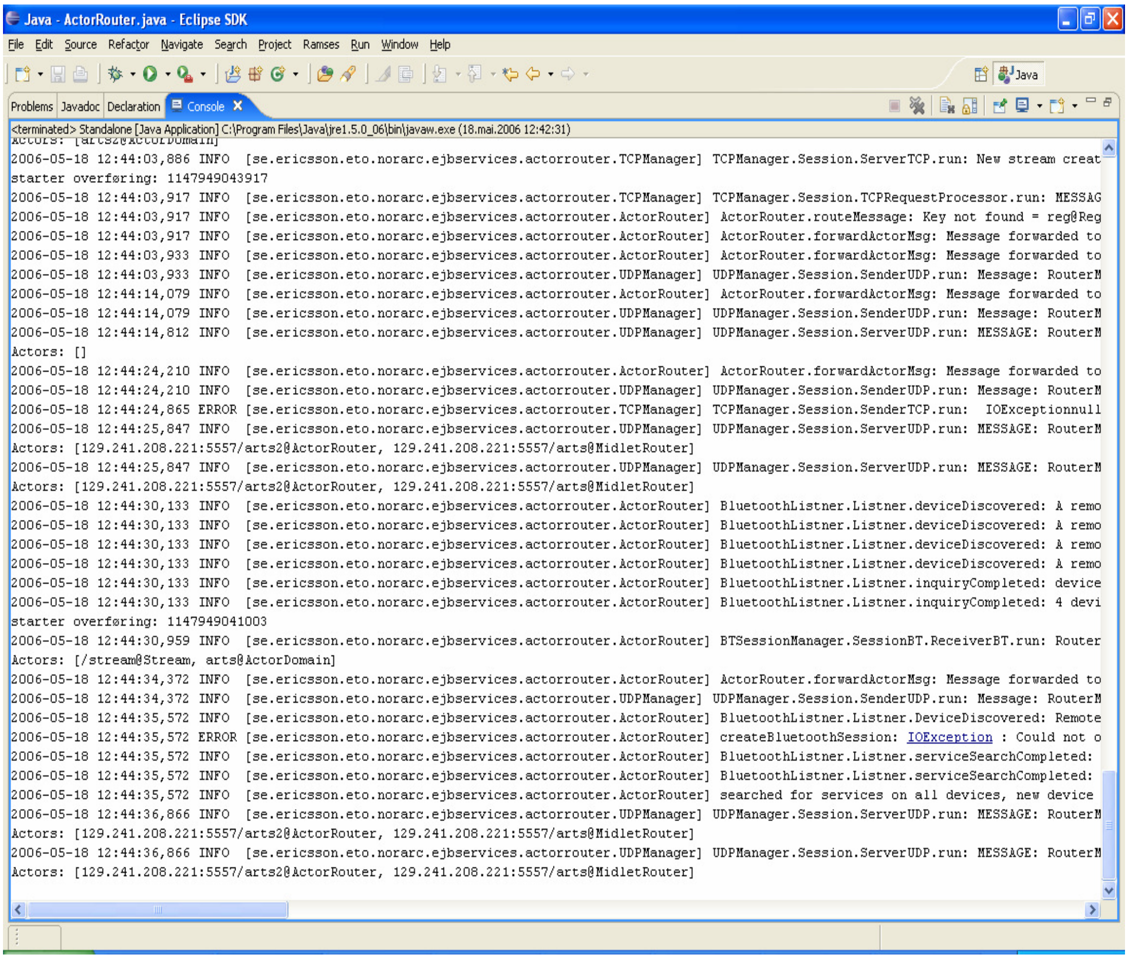


Figure E-4: Printouts available on device (2) (figure 31, chapter 6.2.1)

Printouts from the second desktop computer (device(3) in figure 31, chapter 6.2.1):

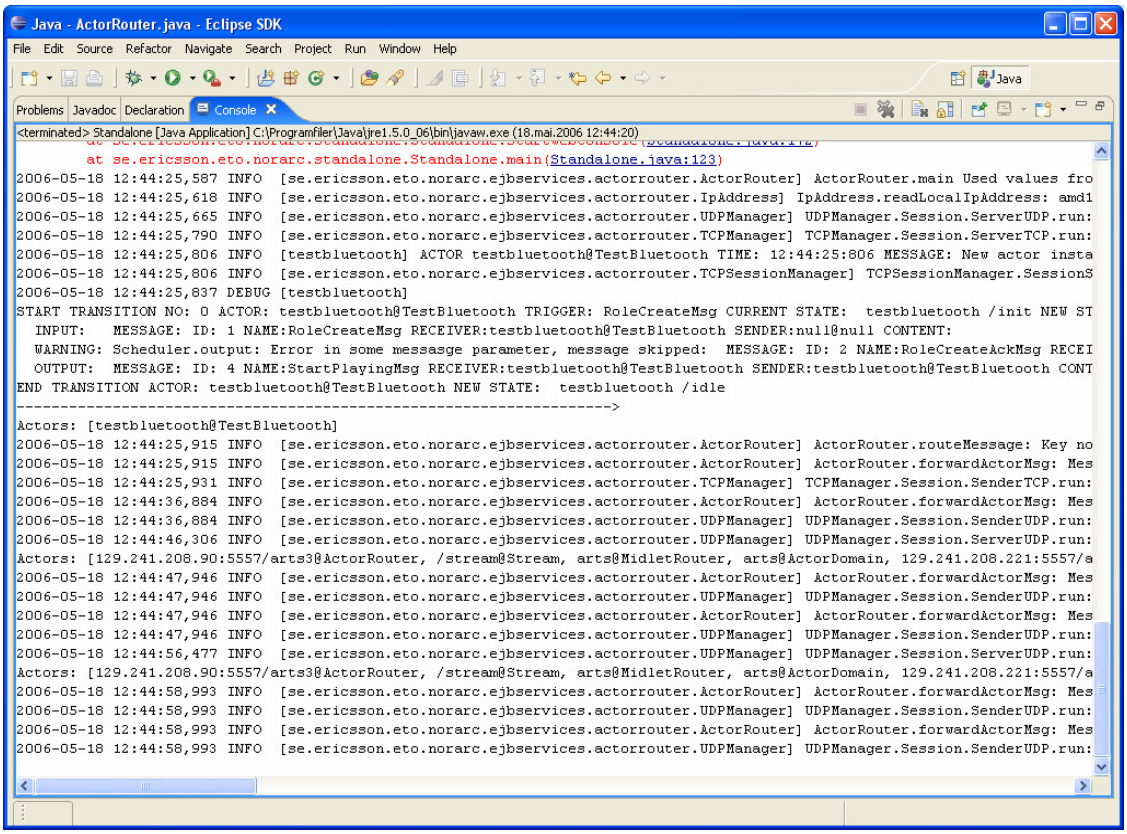


Figure E-5: Printouts available on device (3) (figure 32)

Appendix F: Calculations in MediaClass.java and RouterMsg.java

The calculations are really quite simple, and have been made valid only if the mp3 streamed is split in two.

Calculation of average transfer speed (MediaClass.java):

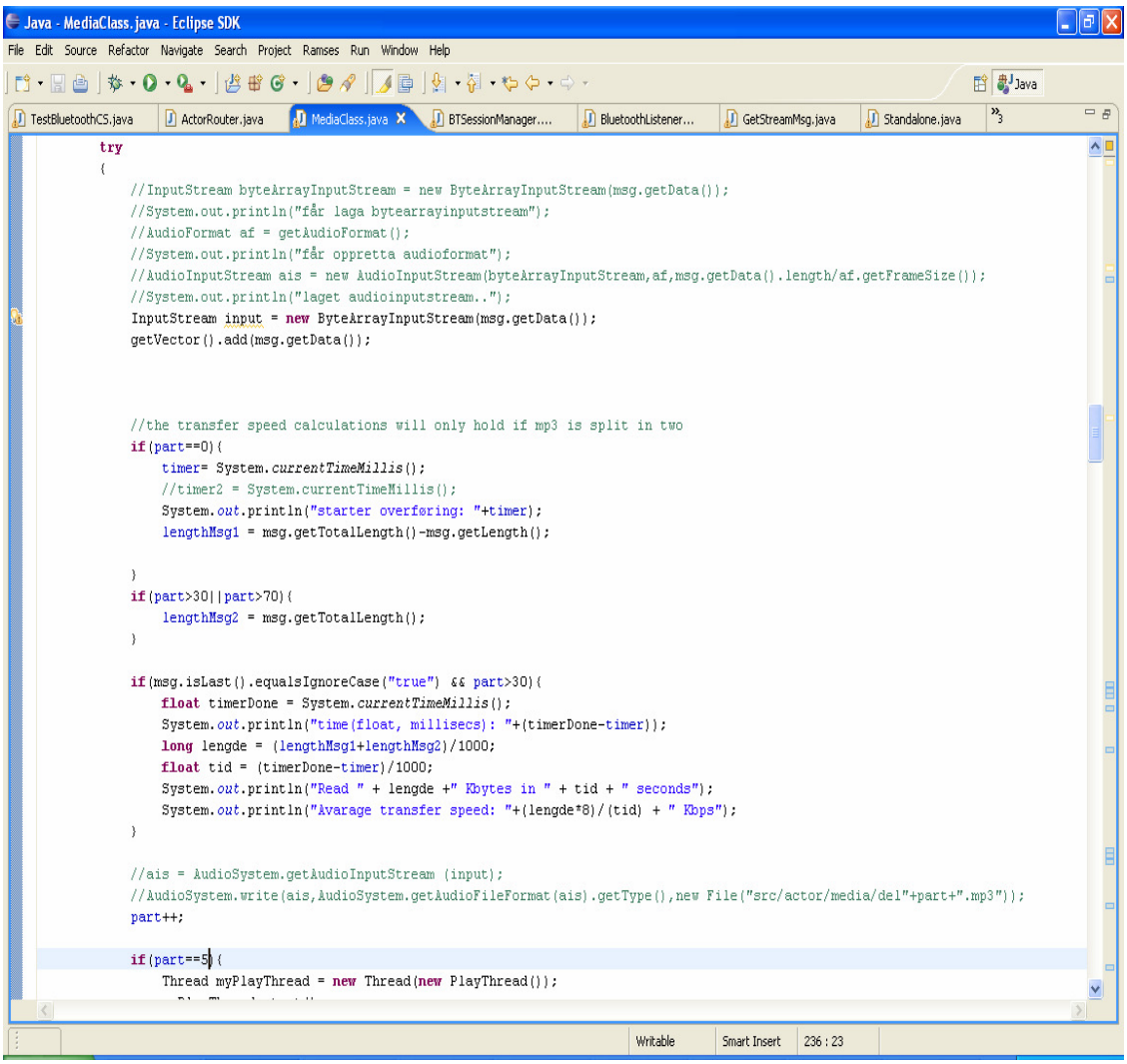


Figure F-1: The calculations of average transfer speed in MediaClass.java (only in the ActorFrame package)

As can be seen time is stored after the first StreamMsg is received and after the last StreamMsg is received. The length of part 1 and part 2 of the split mp3 is also stored. Knowing data length and transfer time allows us to calculate the average transfer speed.

Transfer speed per StreamMsg (RouterMsg.java):

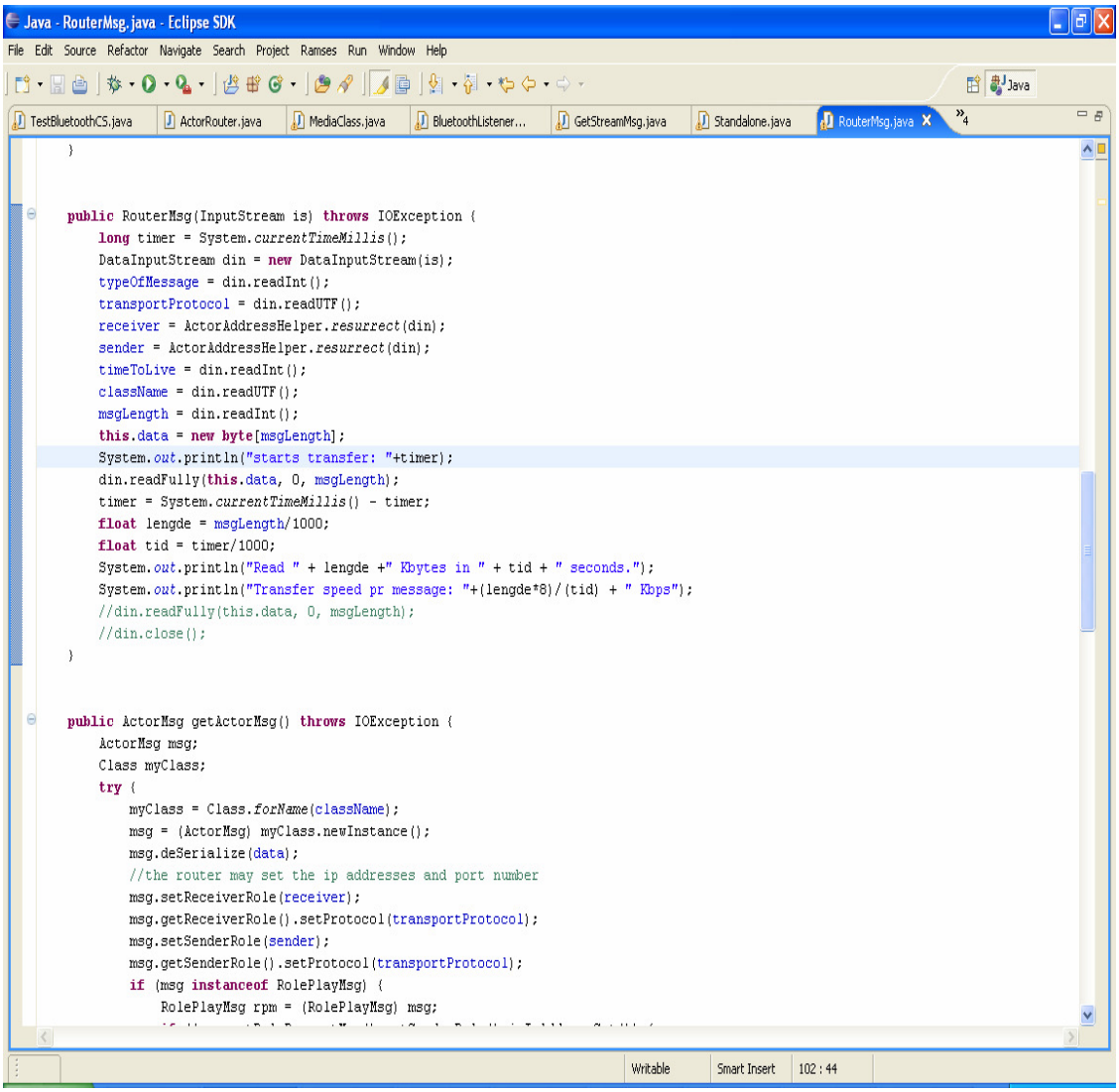
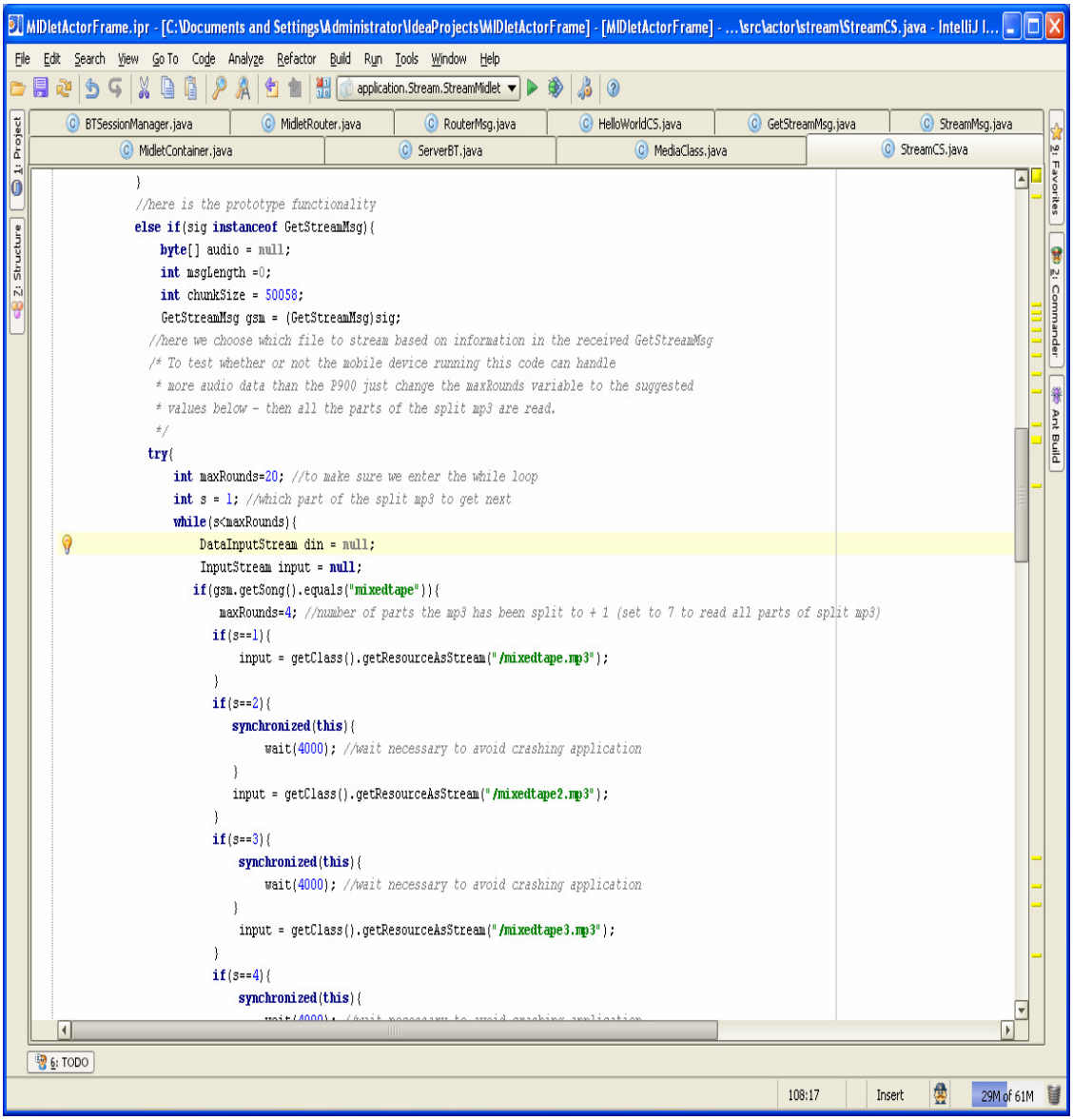


Figure F-2: The calculations of transfer speed per StreamMsg in RouterMsg.java (only in the ActorFrame package)

Here the same principle is used. We know how much data one StreamMsg contains and the transfer time can be calculated. This way the transfer speed is calculated.

Appendix G: Code samples from the prototype

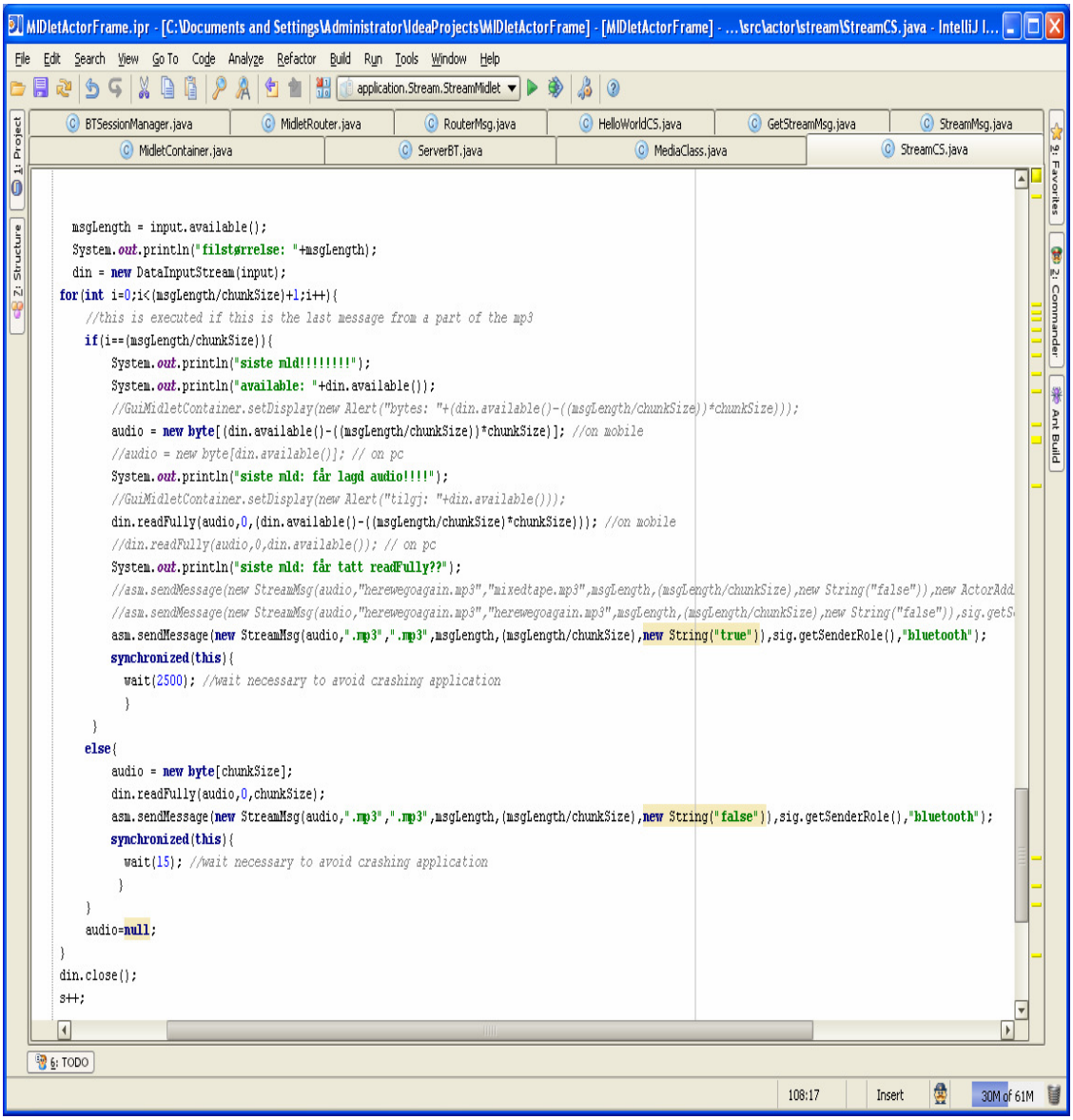
Code from StreamCS.java on the mobile phone; shows how a request for an MP3 is handled:




```
        }
        //here is the prototype functionality
        else if(sig instanceof GetStreamMsg){
            byte[] audio = null;
            int msgLength =0;
            int chunkSize = 50058;
            GetStreamMsg gsm = (GetStreamMsg)sig;
          //here we choose which file to stream based on information in the received GetStreamMsg
          /* To test whether or not the mobile device running this code can handle
           * more audio data than the P900 just change the maxRounds variable to the suggested
           * values below - then all the parts of the split mp3 are read.
           */
          try{
              int maxRounds=20; //to make sure we enter the while loop
              int s = 1; //which part of the split mp3 to get next
              while(s<maxRounds){
                  DataInputStream din = null;
                  InputStream input = null;
                 if(gsm.getSong().equals("mixedtape")){
                     maxRounds=4; //number of parts the mp3 has been split to + 1 (set to 7 to read all parts of split mp3)
                   if(s==1){
                       input = getClass().getResourceAsStream("/mixedtape.mp3");
                   }
                   if(s==2){
                      synchronized(this){
                           wait(4000); //wait necessary to avoid crashing application
                      }
                      input = getClass().getResourceAsStream("/mixedtape2.mp3");
                   }
                   if(s==3){
                       synchronized(this){
                           wait(4000); //wait necessary to avoid crashing application
                       }
                       input = getClass().getResourceAsStream("/mixedtape3.mp3");
                   }
                   if(s==4){
                       synchronized(this){
```

Figure G-1: Screenshot shows instantiation of variables and how a part of the mp3 is read into an InputStream

Here instantiation of variables is shown. In addition it is shown that some wait periods had to be put in to avoid the prototype application from crashing. Also how to read a part of the mp3 into an InputStream is shown.




```
  msgLength = input.available();
  System.out.println("filstørrelse: "+msgLength);
  din = new DataInputStream(input);
for(int i=0;i<(msgLength/chunkSize)+1;i++){
   //this is executed if this is the last message from a part of the mp3
   if(i==(msgLength/chunkSize)){
       System.out.println("siste mld!!!!!!!!");
       System.out.println("available: "+din.available());
       //GuiMidletContainer.setDisplay(new Alert("bytes: "+(din.available()-((msgLength/chunkSize))*chunkSize)));
       audio = new byte[(din.available()-((msgLength/chunkSize))*chunkSize)]; //on mobile
       //audio = new byte[din.available()]; // on pc
       System.out.println("siste mld: får lagd audio!!!!");
       //GuiMidletContainer.setDisplay(new Alert("tilgj: "+din.available()));
       din.readFully(audio,0,(din.available()-((msgLength/chunkSize)*chunkSize))); //on mobile
       //din.readFully(audio,0,din.available()); // on pc
       System.out.println("siste mld: får tatt readFully??");
       //asm.sendMessage(new StreamMsg(audio,"herewegoagain.mp3","mixedtape.mp3",msgLength,(msgLength/chunkSize),new String("false")),new ActorAdd
       //asm.sendMessage(new StreamMsg(audio,"herewegoagain.mp3","herewegoagain.mp3",msgLength,(msgLength/chunkSize),new String("false")),sig.getS
       asm.sendMessage(new StreamMsg(audio,".mp3",".mp3",msgLength,(msgLength/chunkSize),new String("true")),sig.getSenderRole(),"bluetooth");
       synchronized(this){
         wait(2500); //wait necessary to avoid crashing application
         }
    }
   else{
       audio = new byte[chunkSize];
       din.readFully(audio,0,chunkSize);
       asm.sendMessage(new StreamMsg(audio,".mp3",".mp3",msgLength,(msgLength/chunkSize),new String("false")),sig.getSenderRole(),"bluetooth");
       synchronized(this){
         wait(15); //wait necessary to avoid crashing application
         }
   }
   audio=null;
}
din.close();
s++;
```

Figure G-2: Screenshot shows how messages audio data is put into StreamMsg's and sent to the requestor

Normally audio data the size of the defined chunk size is read into the audio byte array, and thereafter put into a StreamMsg and sent to the requestor. If there is less than the chunk size available (the rest of the file has been sent) one must take this into consideration.

Code from TestBluetoothCS.java and MediaClass.java (with inner class PlayThread) on the desktop computer:

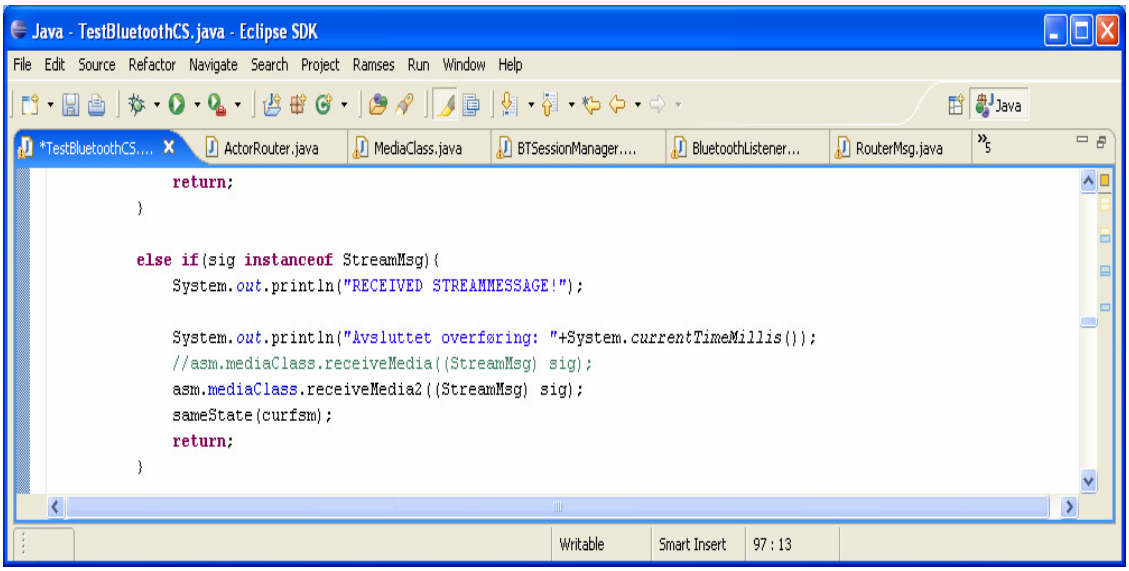


Figure G-3: The StreamMsg's are received and handled by the MediaClass (from TestBluetoothCS.java)

Shows how the StreamMsg's are received and how they are passed on to the MediaClass.

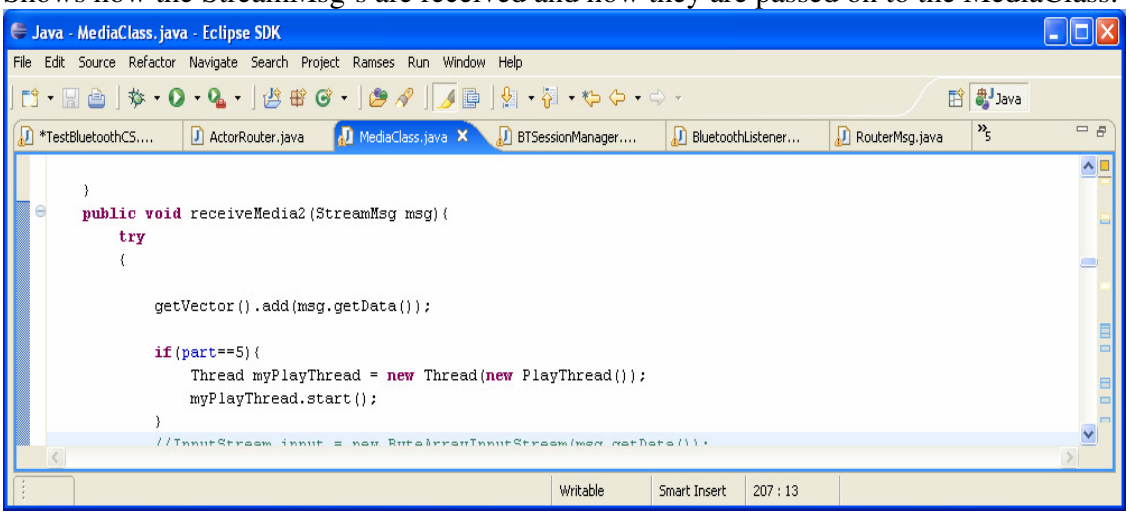


Figure G-4: Messages are added to the buffer vector (from PlayThread, inner class of MediaClass.java)

After receiving the StreamMsg it is added to the buffer vector. If a big enough buffer is ready one can start to play back the mp3.

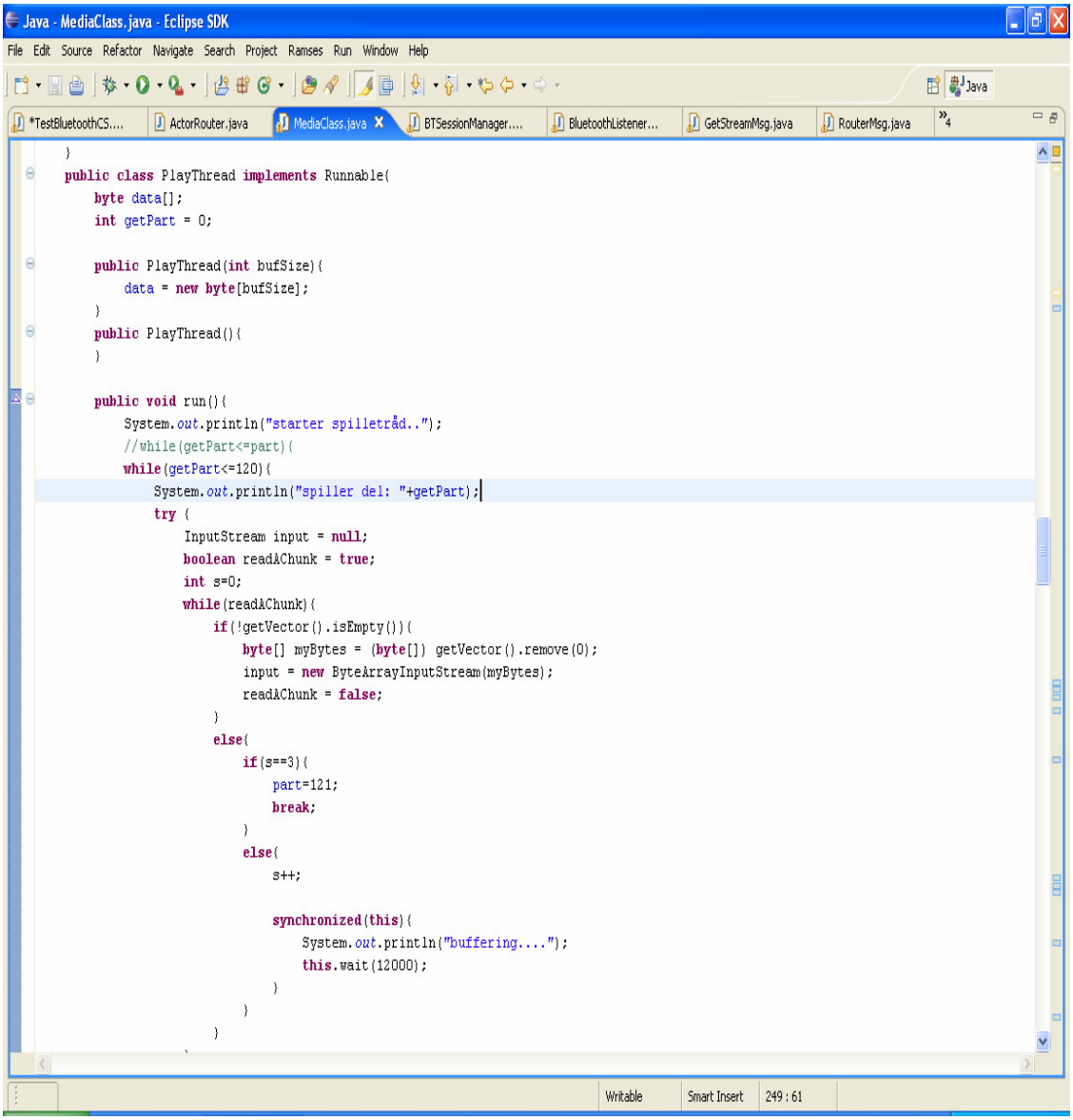


Figure G-5: Must retrieve audio data from the buffer vector. If no audio data – buffering is done (from PlayThread, inner class of MediaClass.java)

When starting playback one must retrieve the audio data from the buffer vector and make an InputStream. If the buffer vector does not contain any audio data buffering is done.

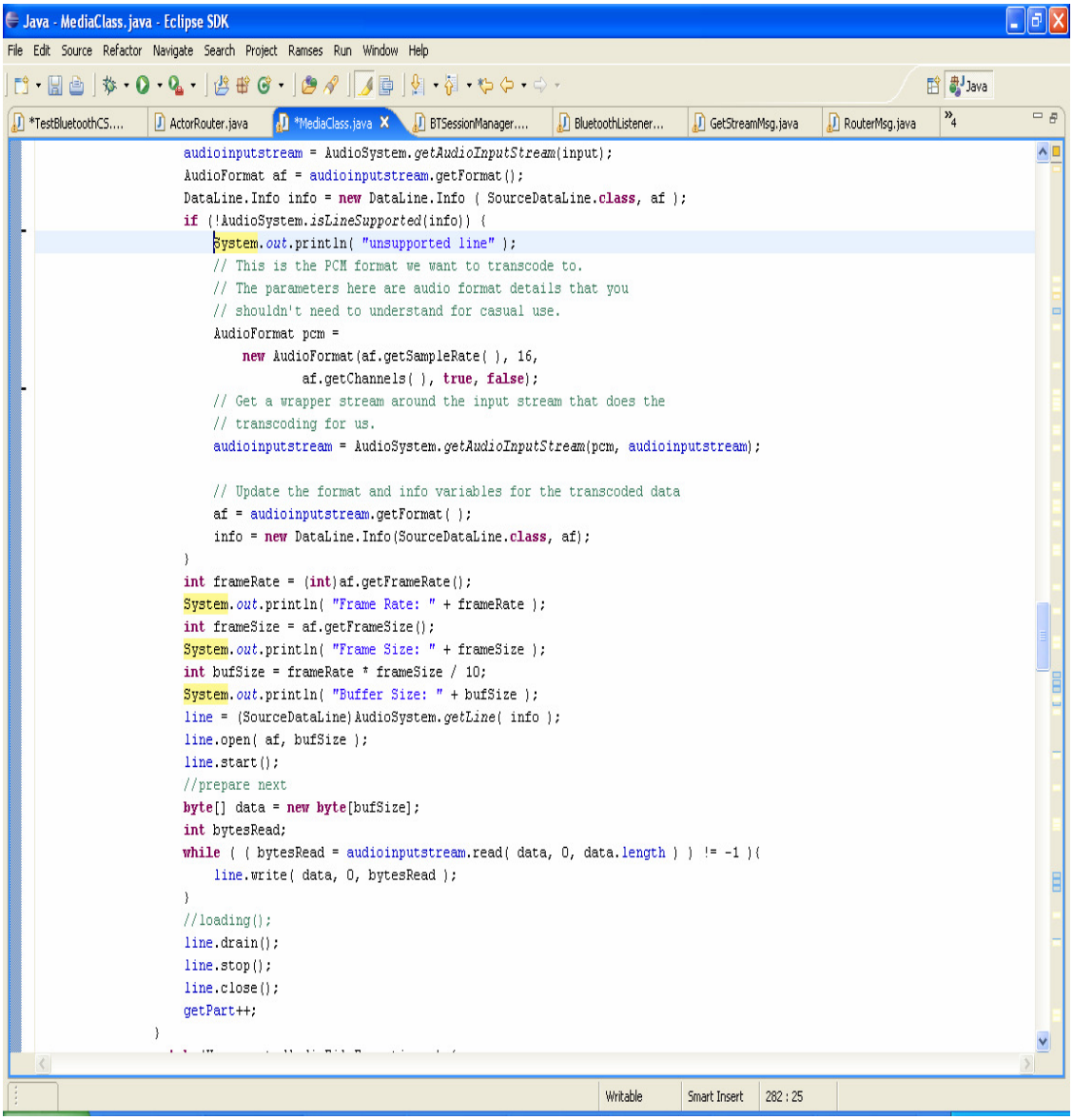


Figure G-7: After audio data is retrieved and read into an InputStream the InputStream is wrapped into an AudioInputStream. (from MediaClass.java)

This screenshot shows how one wraps the InputStream from the previous screenshot into an AudioInputStream and converts the audio format if it is unknown. Thereafter some operations are done and one is ready to play back the audio data.

Appendix H: Attachments included on CD

- EclipseRouting.zip – Needed to run a desktop computer purely as a router
- EclipseStreaming.zip – Includes configurations that enables the desktop computer to stream mp3 files
- The MP3 files – not split
- References – references used in the thesis
- commonLib - .jar files necessary to run ActorFrame
- mp3support - .jar files necessary to enable mp3 support in J2SE
- intellbth.dll - .dll file necessary to enable Bluetooth
- mp3split – trial version of "Easy MP3 split"; a software used to split mp3's to several parts