Andreas Weibel, Lukas Winterhalter

Bluetella: File Sharing for Bluetooth Enabled Mobile Phones

Student Thesis SA-2005-3
Winter Term 2004/05

Tutors:
Matthias Bossardt, Vincent Lenders, Dr. Martin May

Supervisor:
Prof. Dr. Bernhard Plattner

9.2.2005
Abstract

This semester thesis examines the potential and the limitations of the Bluetooth functionality, which the latest generation of mobile phones offers to the Java™ 2 Platform, Micro Edition (J2ME). A J2ME MIDlet - called Bluetella - is described that implements a file sharing application. This application enables the user to search and download any type of file from neighbouring peers. The MIDlet uses only standard Java APIs. Communication and data exchange is done utilizing Bluetooth connectivity. The operation of Bluetella was tested successfully on emulators as well as on real phones.

In contrast to most other file sharing protocols, the current version of Bluetella does not support any routing algorithm. This originates from the incapability of the Bluetooth hardware of today’s mobile phones to build up network-like communication structures beyond point-to-multipoint connections. In order to sidestep this handicap, a concept is shaped and implemented which spreads popular files by making use of the mobility of the human user of the mobile phones.
# Contents

1 Introduction .......................... 1  
1.1 Motivation .......................... 1  
  1.1.1 File Sharing ...................... 1  
1.2 How to Read This Report .......... 2  
1.3 The Already Existing Bluetella Application ........... 2  
1.4 Task Description .................. 2

2 Bluetooth and Java ................. 5  
2.1 Bluetooth .......................... 5  
  2.1.1 Communication Layers .......... 5  
  2.1.2 Piconet and Scatternet ......... 6  
2.2 Java on Mobile Phones ............. 7  
  2.2.1 APIs Used in Bluetella ........ 8  
  2.2.2 Version Numbers ............... 9

3 Development Platform .............. 11  
3.1 Requirements .................... 11  
  3.1.1 Mobile Phone .................. 11  
  3.1.2 Emulator ...................... 11  
3.2 Mobile Phone Manufacturer ........ 12  
  3.2.1 Siemens ...................... 12  
  3.2.2 Motorola ..................... 12  
  3.2.3 Sony-Ericsson ................. 12  
  3.2.4 Sagem ....................... 13  
  3.2.5 Nokia ....................... 13  
3.3 Nokia Mobile Phones ............ 13  
  3.3.1 Developer Platform Series .... 13  
  3.3.2 Nokia 6630 .................... 13  
3.4 Development Environment .......... 14  
  3.4.1 Sun J2ME Wireless ToolKit .... 14  
  3.4.2 Sun Java Studio Mobility 6 ... 15  
  3.4.3 NetBeans 4.0 .................. 15

4 Concepts ........................... 17  
4.1 Searching Files ................... 17  
  4.1.1 Why a Complex Search System? .. 17
7 Test Results

7.1 Functional Tests
  7.1.1 Discovering Peers
  7.1.2 Correctness of the Results
  7.1.3 Most Wanted List
  7.1.4 Downloading
  7.1.5 More than One Download
  7.1.6 Interrupted Downloads
  7.1.7 User Interface
  7.1.8 Energy Consumption

7.2 Performance Tests
  7.2.1 Discovering Peers
  7.2.2 Creating the MD5 Hash
  7.2.3 Download Speed
  7.2.4 Memory Usage

8 Conclusion

8.1 Summary
8.2 Future Work
  8.2.1 Signing the Application
  8.2.2 Additional Features
  8.2.3 Most Wanted List
  8.2.4 Security
  8.2.5 Other Devices
  8.2.6 From RFCOMM to L2CAP
  8.2.7 Is Bluetooth Really the Way to Go?

A Bluetella User Manual

A.1 Installation
A.2 Menu Navigation
  A.2.1 Search Menu
  A.2.2 Results Menu
  A.2.3 Transfer Menu
  A.2.4 Info Menu
  A.2.5 Preferences Menu

B CD Contents

C User Interface

D Known Issues
  D.1 Connectivity Limitations
  D.2 Stability
  D.3 Shutdown Duration
Chapter 1

Introduction

1.1 Motivation

In the last few years, the world of telecommunication has changed dramatically. A decade ago, no one deemed it possible that today almost everybody uses mobile phones and broadband Internet access. Local wireless networks are mushrooming, the third generation of mobile phones is ready for market and with Internet telephony, the traditional telecommunication companies face competition even in their core business.

Yet, the evolution is far from being at an end point. Communication will become even more mobile, cheaper and more decentralized. Ad-hoc or mesh networks [3] probably will turn the telecommunication market upside down once more. It’s not even clear, whether future mobile networks will still work on a commercial basis or in a participatory, peer-to-peer-like manner [4].

Bluetooth with its short radio range certainly is not the technology that will scare the telecommunication industry. Nevertheless, it is an interesting model to try out new concepts, in our case a wireless file sharing network that makes use of the mobility of its human users.

1.1.1 File Sharing

What kind of files could be shared with Bluetella? The first idea that one will come up with are music files and ringtones. This is the most common use of present file sharing applications and probably would not be different with Bluetella. However, Bluetella could also be applied for other benefits. For example to make sure that in disaster situations, the emergency personnel can still access and update files, although central communication facilities have failed.

Of course, file sharing of media data entails a number of legal issues. Since we are not experts in this area, we will not address this topic further in this thesis. It is clear that Bluetella could be used to exchange files without having the permission to do so. Yet, this is also possible using other programs that are delivered with the phones.
1.2 How to Read This Report

In the first chapter, the predecessor Bluetella semester thesis [1] will be introduced, and the precise task given to us will be described. In the second chapter, the most important basics on the used technologies will be summarized. In the third chapter, the choice of the used phones and development tools is explained. In the fourth chapter, the concepts of Bluetella are depicted. The fifth chapter contains the specification of the communication protocol designed for Bluetella. The sixth chapter will explain the actual implementation of the concepts. The seventh chapter describes the tests we carried out with Bluetella and their results. And finally, the findings of this thesis will be summarized in the eighth chapter.

To learn more on the “communication by mobility” concept, particularly Chapter 4 is useful. Technical information on Bluetooth for J2ME is mainly found in Chapters 2, 3 and 7. Chapters 5 and 6 probably will be of greater interest only for persons who want to continue the work on Bluetella.

1.3 The Already Existing Bluetella Application

In the summer term of 2003, there has been a semester thesis named “Bluetella: A Java Application for New Mobile Phones” [1]. It was written by Ganymed Stanek at the Computer Engineering and Networks Laboratory (TIK) at ETH. While the idea behind Stanek’s thesis was quite the same like for this thesis, the actual implementations differ more or less completely. This is mainly because many things have changed since 2003. First of all, no phone supported JSR-82 at the time the first Bluetella thesis was written. Hence, there was no chance for Stanek to test his Bluetella on real phones, so he could only use an emulator. At that time, is was also unknown that Bluetooth connection establishment takes such a long time with the kind of devices used for Bluetella. For that, Stanek’s implementation included a multi-hop, ad-hoc source routing protocol. As described in Section 4.2, we decided to use a different approach to spread files across the network. Another major difference is the implementation of the file access: The J2ME standard for access to file systems was released only in June 2004. Therefore, Stanek was forced to use a manufacturer-specific library. It was also for this reason, that he decided in favor of the Siemens SL55 emulator as platform to test his application.

1.4 Task Description

Our goal was to design and implement a working prototype of a Bluetooth file sharing application. On the basis of the already existing Bluetella version, we had to create an application that works on real mobile phones instead of running only in an emulator. In order to guarantee the portability of Bluetella, we had to use solely standardized APIs, wherever possible.
Bluetella is part of the Blue* [BlueStar] project [2].

The tasks we were given were:

- Get familiar with the JVM and related APIs provided by the latest models of mobile phones. In particular study the Bluetooth API
- Get familiar with the former semester thesis on Bluetella
- Identify mobile phones with Bluetooth support which are suitable for the Bluetella application. Categorize different models based on a set of chosen characteristics. This task should be done in collaboration with the participants of the “BlueDating” thesis.
- Propose different technical approaches for the file sharing application and compare them with each other. Select the most promising alternative.
- Design and implement the protocols for the chosen alternative.
- Define and set up a demonstration scenario for Bluetella. Preferably the demonstration should run on currently available mobile phones.
- Document your work in a detailed and comprehensive way. New concepts and investigated variants must be described. Decisions for a particular variant must be justified.

For the entire assignment, please refer to Appendix E.
Introduction
Chapter 2

Bluetooth and Java

2.1 Bluetooth

Bluetooth is a short-range radio technology designed for wireless connectivity between mobile devices. It was publicly announced by the Bluetooth Special Interest Group\(^1\) in early 1998. Since then, it has become more and more the de-facto standard in wireless connectivity. It should not be compared directly to WLAN\(^2\) that uses the same frequency range. Bluetooth was designed as a short-range cable replacement optimized for small size, minimal power consumption, and low price. In contrast, WLAN was designed for faster data rates and greater transmission ranges requiring more expensive hardware.

For detailed information on Bluetooth technology, we recommend the following readings: [5], [6], [7], and [8].

2.1.1 Communication Layers

Like every communication system, Bluetooth technology is divided into a number of layers, each using functionality of the lower layer and providing some functions to the upper layer.

**Baseband** The lowest layer of the Bluetooth stack are the physical radio and Baseband layers. These layers control the Bluetooth unit’s synchronisation and transmission frequency hopping sequence.

**LMP** The Link Manager Protocol is responsible for link setup between Bluetooth units. It handles the control and negotiation of packet sizes used when transmitting data.

**HCI** The Host Controller Interface provides a uniform interface to the Bluetooth hardware (Radio, Baseband and LMP).

---

\(^1\)The Bluetooth Special Interest Group (SIG) was founded by Sony Ericsson, IBM, Intel, Toshiba and Nokia.

\(^2\)We will subsume all technologies based onto the IEEE standard 802.11 by the term WLAN (Wireless Local Area Network).
**L2CAP** The *Logical Link Control and Adaptation Layer Protocol* guarantees reliable transmission and provides protocol/channel multiplexing. L2CAP uses the *Asynchronous Connection-Oriented (ACL*) link type defined by Bluetooth. This link type supports symmetric and asymmetric channels. The symmetric channel can transfer data with 433.9 kilobits per second in both directions (optimal case). The asymmetric channel allows an increase of the data rate up to 721 kbit/s in the forward direction at the cost of decreasing the reverse link speed to 57.6 kbit/s. There is quite a number of protocols that are set upon L2CAP such as RFCOMM and SDP.

**SDP** The *Service Discovery Protocol* allows to search for certain services on a remote Bluetooth device.

**RFCOMM** The *Radio Frequency Communications Protocol* emulates an RS-232 serial connection between two Bluetooth devices. One limitation is that two devices can share only one RFCOMM link at the same time. However, more than one logical serial connection can be multiplexed over this session. A RFCOMM session is uniquely identified by the Bluetooth addresses of its endpoints.

### 2.1.2 Piconet and Scatternet

A Bluetooth network is called *piconet*. In the simplest case, this means that two devices are connected (Fig. 2.2a). The device which controls
the piconet is called *master* and the other participating devices are called *slaves*. Normally, the master is also the device that initiated the connection. A master can have several simultaneous connections (point-to-multipoint) up to seven slaves (Figure 2.2b). The master/slave roles are not necessarily fixed by definition, but very few devices support switching the role during the connection. Nokia’s phones do not support this feature neither.

![Figure 2.2: Bluetooth piconet and scatternet scenarios: a) Point-to-point connection between two devices; b) Point-to-multipoint connection between a master and three slaves; c) Scatternet that consists of three piconets](image)

In a *scatternet*, one device can also be connected in two or more piconets (see Fig. 2.2c). A device can, however, only be a master to one piconet at a time. Most of the current Bluetooth implementations support piconets only [6]. Actually, at the time this thesis was written, no mobile phone was known to feature scatternets at all. This drawback will be crucial for the decision if and how a routing functionality should be part of Bluetella.

### 2.2 Java on Mobile Phones

In 1999, Sun Microsystem announced the release of a new edition of the Java family called *Java 2 Micro Edition (J2ME)*. Figure 2.3 gives an overview. This platform was aimed at the growing market of mobile and embedded devices [10]. Two different *configurations* were defined, the *Connection Device Configuration (CDC)* and the *Connection Limited Device Configuration (CLDC)*. While CDC targeted relatively powerful devices, CLDC approached smaller devices like low to mid-end mobile phones. Up to now, CDC has enjoyed very little success on the consumer market, whereas J2ME CLDC today is implemented on a wide range of mobile phones.

In order to make Java even more customizable for the needs of different appliances, an additional layer, called *profile*, was defined on top of the
configuration. This layer should integrate the functionality of one group of devices. Mobile phones use the Mobile Information Device Profile (MIDP).

On top of the profiles, different optional packages can be used. These packages contain special standardized APIs. Additionally, most phone manufacturers provide specific APIs for some phones or classes of phones. [1]

![Figure 2.3: Overview of the various Java versions](image)

### 2.2.1 APIs Used in Bluetella

On top of the above mentioned profiles, Java devices can support a number of additional APIs. For Bluetella, the following APIs were used:

**Bluetooth API:** The Bluetooth API is specified in JSR-82 [12]. It provides functionality to find and connect to other Bluetooth devices and to exchange data with them.

**File Connection API:** This API originally was designed for PDAs. It offers a convenient access to the file system of the device. The File Connection API is specified in JSR-75 [13].

**MIDP and CLDC:** The MIDP and CLDC APIs contain the general functionality used in Bluetella like user interface, stream connections or record stores (see Section 4.4). They are specified in JSR-118 [14] and JSR-139 [15], respectively.
To provide as much portability as possible, no manufacturer specific API was used. As Nokia’s API [16] does not contain any feature of big importance for Bluetella, this was not a great sacrifice.

2.2.2 Version Numbers

At the time of the writing, the following Java versions were the current ones:

**MIDP 2.0**: Version 2 of MIDP provides a number of improvements in comparison with previous versions. Among them are: Enhanced features to build a portable graphical user interface, more support for media applications and improved network connectivity. In particular, MIDP 2.0 adds support for HTTPS and UDP connections as well as serial port communication.

**CLDC 1.1**: Version 1.1 brings only minor enhancements to the previous version 1.0.4 that make it more similar to J2SE, such as floating point and weak references support.
Bluetooth and Java
Chapter 3

Development Platform

In order to test Bluetella, it was necessary to use a mobile phone. In the following sections we will give a brief overview over the market at the time of writing of this thesis and describe what possibilities we had to choose from. Furthermore, we will describe the different development environments we used.

3.1 Requirements

Based on the information given in Section 2.2, the requirements for the mobile phone and the corresponding emulator are summarized here. As stated later in Section 3.4.1, Sun’s J2ME Wireless Toolkit (WTK) [19] was used for all emulating purposes. However, when we started our thesis, the WTK did not yet support Bluetooth emulation. Therefore, we had to take into account the availability of an emulator coming from the manufacturer for our mobile phone choice. Sun’s WTK 2.2 actually made this requirement obsolete after the mobile phone had been already chosen (see Section 3.2).

In the next two subsections, we summarize the requirements for the mobile phone and the emulator (as stated in the beginning of our thesis).

3.1.1 Mobile Phone

- **MIDP 1.0**: For Java support. Actually, all newer Java phones support even the newer MIDP 2.0. However, MIDP 2.0 is nice to have but not a crucial requirement.

- **JSR-82**: Fundamental for Bluetooth connectivity.

- **JSR-75** Necessary for the access of the file system beyond the application’s record store (cf. Section 2.2.1).

3.1.2 Emulator

- The emulator clearly should provide support for both JSR-82 Bluetooth communication and JSR-75 file access. JSR-82 support with a working Bluetooth stack is crucial. With a missing JSR-75 we could
cope somehow and only use the record store functionality provided by MIDP when emulating.

- It should be possible to run multiple phones simultaneously having them communicating via Bluetooth.

- A possibility to influence the communication of the emulated phones, that is to simulate distance/visibility between the phones would be helpful.

- Finally, the emulator should be available at a reasonable cost.

3.2 Mobile Phone Manufacturer

After framing the requirements, we researched which suppliers of mobile phones did meet the demands. In the following, the results of these investigations are listed:

3.2.1 Siemens

Siemens had several Java Bluetooth enabled phones on sale. None of their phones had JSR-75, instead they used their proprietary Siemens PIM API, which - among other things - supports file access.

Siemens provides its own SDK with several available emulator packs. Unfortunately, it was not yet possible to run more than one emulator at the same time with this software. Thus, we were not able to simulate the communication between mobile phones. A second drawback was, that Siemens had no emulator support for its newest models.

In his predecessor thesis [1], Ganymed Stanek used the Siemens SL55 emulator pack combined with the Rococo Impronto Simulator. We decided not to continue on this way since, first, no emulators for the recently released JSR-82 mobiles are available from Siemens and secondly, there is no student or evaluation license available for the Rococo Impronto Simulator\(^1\) anymore.

3.2.2 Motorola

Motorola is quite active in the Bluetooth market. However, their mobiles currently do not yet support JSR-82.

3.2.3 Sony-Ericsson

There exist several mobiles from Sony-Ericsson, that would fit in our requirements, namely the P900, P910 series. Unfortunately, the available Sony-Ericsson emulator did not support Bluetooth.

\(^1\)The Rococo Impronto Simulator integrates the Siemens emulator and enriches it with several features. One of them is the simulation of Bluetooth communication between a number of mobile phones. However, this product costs now 1000€ per developer.
3.3 Nokia Mobile Phones

3.2.4 Sagem

To the best of our knowledge, no Sagem phones do support JSR-82 at the moment.

3.2.5 Nokia

Nokia does offer some phones that are compliant with the JSR-82 and JSR-75 standards. Furthermore, Nokia was the only manufacturer who provided a serviceable emulator that can simulate the Bluetooth communication of multiple mobile phones. Additionally, the emulator can be used in standard development environments (see Section 3.4). Therefore, Nokia was the option we finally chose.

3.3 Nokia Mobile Phones

In this section, we will give a brief overview over the available phones of this manufacturer in order to choose one for our thesis.

The JSR-82 API is available on all newer Bluetooth enabled Nokia phones with Java. However, only a few of them support also the JSR-75 API, which turned out to be the most restricting criteria for the selection.

3.3.1 Developer Platform Series

Nokia phones are categorized in four Developer Platform Series. Each developer platform is designed to provide developers with common APIs across a range of devices. [20]

**Series 40 Developer Platform:** Java technology-enabled mobile devices for the mass market. Based on Nokia’s OS.

**Series 60 Developer Platform:** Smartphones, with Java technology capability. Based on SymbianOS.

**Series 80 Developer Platform:** Enterprise devices. Based on SymbianOS.

**Series 90 Developer Platform:** Media devices. Based on SymbianOS.

As it was clear that a normal mobile phone should be used for Bluetella, series 80 and 90 phones dropped out.

3.3.2 Nokia 6630

At the time this thesis was started, there were only two Nokia phones available that supported JSR-82 as well as JSR-75: Nokia 9300 and Nokia 6630. As the 9300 model belongs to the platform series 80, the 6630 was the only device that met our criteria\(^2\).

\(^2\)In the meanwhile, a few more models have become available which support both of these APIs. Among them are the 7270, the 7280 and the 6255
The 6630 model was announced in June 2004. Some of its features are:

- 176 x 208 pixel screen with 16 bit color depth
- 10 MB memory with the option to use additional memory cards
- CLDC 1.1, MIDP 2.0 with JSR-82 and JSR-75
- An integrated 1.3 megapixel camera with 6x digital zoom
- Support for GSM, GPRS, EGPRS, W-CDMA (i.e. UMTS)

For more information on the 6630, please refer to [21].

### 3.4 Development Environment

The main criteria for a development environment was support for J2ME MIDP developing and that it can integrate the above chosen emulator. Searching only for freely available software we mainly found two possibilities. Both, the Netbeans IDE 4.0 as well as the Sun One Studio Mobility 6 met our requirements. Because the final release of Netbeans 4.0 outrivaled Sun’s Mobility Studio, we used the former for coding, emulating and debugging.

#### 3.4.1 Sun J2ME Wireless ToolKit

The Sun J2ME Wireless Toolkit [19] (WTK) is a toolbox for developing wireless applications that are based on J2ME, CLDC and MIDP. It includes an emulation environment with a couple of standard emulators. With the WTK it is possible to compile Java source code for mobile devices and test it in an emulator. During the writing of the thesis, Sun released version 2.1 and version 2.2 of the WTK. The latter turned out to be a fairly powerful tool, providing an emulator with support for JSR-75 as well as for Bluetooth connectivity.
3.4 Development Environment

3.4.2 Sun Java Studio Mobility 6

The Sun Java Studio Mobility 6 [22] is an IDE for developing applications that can be deployed to Java technology-enabled mobile devices. It is built upon Netbeans 3.6 with additional features included by Sun. It comes already bundled with the above mentioned WTK. It can be used at no cost for 60 days.

3.4.3 NetBeans 4.0

Netbeans [23] is an open source IDE for Java developers funded by Sun Microsystems. Since version 4.0, a mobility pack is available that integrates the Sun J2ME Wireless Toolkit. While the beta versions of Netbeans 4.0 showed some stability problems, the final release version demonstrated to be a useful and recommendable development environment.

Figure 3.2: Screen shot of the Netbeans IDE 4.0 running Bluetella in an emulator.
Chapter 4

Concepts

4.1 Searching Files

4.1.1 Why a Complex Search System?

First, we reasoned that an average user does not want to go through a complex search procedure when she has to enter the request into her mobile phone. Therefore, we intended to keep the search procedure extremely simple by allowing only searches for one string, which is then looked for in all available file information.

However, we changed our minds and decided that the user should also be able to fill out a more complex search mask, in which more sophisticated queries would be possible as well. We did this, because we wanted to give the user the possibility to choose the search complexity that she wants to use and, on the other hand, because the storage capacity of mobile phones is increasing constantly and probably will reach the size of portable MP3 players quite soon. Hence, simple queries could easily trigger too many results. With this search mask – to give an example – the user should be able to look for a certain song performed by a certain artist that is encoded with Ogg Vorbis at a minimum average bit rate.

Figure 4.1: Screen shot: Simple search mask
4.1.2 How a User Searches for Files

The user can either do a simple search, only posting one string, which then is looked for in all available file information, or she can specify a list of attributes that a file must meet. The list of this attributes can be seen in Table 5.14. The file type attribute is special as it is the only attribute whose value is not defined by the file itself. For this reason, the categories need to be clear among all Bluetella clients. The categories are listed in Table 5.15. In the search evaluation, all attributes are linked with a logical AND relation. To narrow the search further, it is possible to logically negate an attribute search and to use greater than or equal than operators for certain attributes.

4.2 Spreading Files

As described in Section 2.1.2, it is not possible to have a Bluetooth connection over more than two hops. Therefore, a persistent multi-hop network is not feasible with the current Bluetooth hardware.

We considered to work around this problem by establishing and cutting connections dynamically (see Figure 4.2). But we had to abandon this approach. The establishment of connections takes too much time, since it is necessary to wait until the configuration of the piconets has settled. So, the forwarding of a result, for example, could take several minutes from sender to recipient. Thus, chances would be high that the sender had already moved out of the range of the device before it even had received a possible reply. In addition, there is no way to engage in the administration of communication at deeper layers with J2ME. Therefore, there is no way of speeding up the establishment process.

4.2.1 Using Human Mobility

Because routing was not possible, we decided to venture a completely different approach. It bases on the assumption that the user of a mobile phone moves around and regularly encounters different other peers. On the occasion of this encounters, they transmit what they are looking for, of course, and exchange the files that they are searching. But beyond that, they also download files that other peers were looking for earlier (see Figure 4.3). Like this, files are replicated and spread among the Bluetella users and the chances are enhanced that, occasionally, the clients will find the files that they are searching.

4.2.2 Practical Challenges

It is not feasible for a Bluetella client to remember all searches that he received in the past. Therefore, it is necessary to restrict the range of such searches. Clearly, requests that were seen from more than one peer should be prioritized and more recent searches should outrank older ones.
4.2 Spreading Files

Figure 4.2: A wants to send a packet to D with the source routing protocol Stanek described in [1]. B is obviously the master of a piconet with A and C and has established connections with both of these nodes. It is not possible that D has a connection to C at the same time, as it is not in the same piconet. In a first step (a), A sends a packet to B. In (b), B forwards the packet to C. Then, in (c), C cannot establish a connection with D. Therefore, it closes the connection with B and becomes master of a new piconet with D. Consequently, C can forward the packet to D.

In practice, Bluetella maintains a list with so-called Most Wanted Searches. We will call this list either Most Wanted List or Top10 List. This list of search requests is processed after the normal searches of the user are transmitted. The assembling of the list is done with the aid of a cache of received searches. How this is realized is explained in Chapter 6.5.

What Should Be Downloaded?

Another important question is, which files should be downloaded when there are results to a Most Wanted Search. First of all, no files are downloaded, if the file already exists on the mobile phone\(^1\). We could imagine that it was possible to use an algorithm that heuristically determines the significance of the remaining results. However, we decided that a result is chosen randomly in these cases, because we considered that such a “significance algorithm” would go beyond the scope of this thesis. We neither wanted to choose them by alphabetical order or alike, as this would favor the distribution of certain files without justification.

We do not limit the number or size of the files that are downloaded from a specific host for a search in the Most Wanted List. Thus, it theoretically

\(^1\)This also applies for manual searches
Figure 4.3: (a) A searches a file. The corresponding request is sent to B. B does not have this file, but remembers that someone was looking for it. (b) B has moved to a different location and meets C. Node B now sends A’s memorized search to C. C does have a matching file and B downloads that file. (c) When B has returned to A, A still searches the same file. This time, B can deliver the file that it had downloaded previously from C.

would be possible for a malicious user to fill the corresponding result list with an arbitrary number of entries if she returns different results on each search run. This will finally fill up the phone’s memory with unwanted data and can be considered as a possible Denial of Service (DoS) attack.

Time-to-Live

There is another important issue we had to face: When a client sends out a Most Wanted Search, his partner will interpret this like a normal, manual search request and update his Most Wanted List accordingly. By itself, this relay effect is a desired feature, because like this, popular searches are spread among the Bluetella peers. However, it also produces a serious problem, when search requests start to cycle indefinitely among a group of Bluetella users.

We inhibit in our implementation that searches in the Most Wanted List are sent back to its senders. But this is not enough, because this alone does not hinder packets from circulating over an infinite number of hops. Therefore, we decided to introduce a time-to-live field into the search packet. The value of this field is decremented every time a search request is relayed (see the protocol definitions in Section 5.3.1). When the time-to-live finally has reached zero, results are still sent back as usual, but it is not used for updating the Most Wanted List anymore. Thus, it is possible to prohibit search requests to go forever from one phone to the next.
4.3 Downloading

We had two different options for the design of the file transfer: The files could be split up into single chunks and be downloaded chunk after chunk or the files could be downloaded as a whole.

4.3.1 Chunks

Working with chunks means, that for downloading, the files are divided into consecutively numbered parts with constant size. When downloading, the client requests not the whole file, but a specific chunk.

The main advantage of this approach is that it is much better apt for other transport technologies besides direct Bluetooth connections, for example for source routing. With chunks, it would also be possible to download parts of the same file from different peers at the same time. However, this would normally also cut in half the available bandwidth for download in the piconet - thus resulting in no real gain of download speed. Nevertheless, one could imagine special scenarios, where downloading from different servers could bring an improvement\(^2\).

The disadvantages are: When the transfer of a chunk is interrupted, there is no other way than to drop the already downloaded data of this chunk. Depending of the size of the chunks, this could result in a waste of capacity. Chunk handling would introduce some overhead in the protocol and requires additional resources to manage the chunks.

4.3.2 Entire Files

Downloading entire files means that the download of a file starts at its first byte and continues monotonically until the last byte.

One advantage of this approach is that interrupted downloads can be resumed exactly at the byte where the download stopped. Furthermore, the implementation is simpler and more resource-saving than the chunk concept.

4.3.3 Decision

We finally elected the second option. We considered that the scenarios, where the chunk concept would be faster, were too rare to compensate the possible losses caused by lost chunks and the additional overhead. However, this estimation is not based on empirical data. Besides, it would be possible to adapt the chosen transfer concept to other communication means where no byte stream is available, either by extending the current Bluetella

\(^2\)For Example: We are downloading a file from a server that has only half of its Bluetooth bandwidth free. We would have more transfer capacity, but can not use it due to the bottleneck at the server. Then, another server with also only half of its bandwidth available appears. With chunks, we now could start to download more chunks of the same file from the newly appearing server.
protocol or by introducing an additional layer that is able to reassemble the packet data.

4.4 File Access

There are two different types of file access in J2ME. In the following, we will shortly explain what they are and what we used them for:

4.4.1 Record Store

Record stores are the standard storage method in MIDP. They are memory areas which are reserved for and managed by one single MIDP MIDlet. An application may have multiple record stores. There is no way for an application to read a record store of another application. The record stores contain a set of binary data elements called *records*. The records are accessed using a numerical ID.

In Bluetella, record stores are used to save the preferences data and the internal list structures containing file and search information.

4.4.2 File Connection (JSR-75)

J2ME itself does not offer any functionality to access the files in the memory of a device. Yet, access to files is crucial for Bluetella. For that reason, the File Connection API [13] is used. It offers all functionality to manage and use files in the file system of a device.

In Bluetella, the File Connection API is used to manage the files that are shared or have been downloaded.

4.5 Identifying Files

Using the name of a file is not a very useful way to identify files uniquely. Files carrying the same name are not necessarily identical and a different file name does not automatically imply that the files are not the same. Besides that, file names do not have a predefined size and tend to be rather lengthy. For these reasons, we decided to allocate a unique hash (cf. Chapter 6.4.1) to every file that is handled by Bluetella. This so-called *file ID* is used to organize the management of results, downloads and uploads. It is also used by a client to check the consistency of a downloaded file.

4.6 Security

All mobile applications carry the danger of becoming a security risk. Therefore, it is important to think about the security implications.

Bluetella is a file sharing application. It would be easy to share malicious code with the current implementation. However, to actually run the code the user would need to execute the dangerous code himself. It is clear that
it is a security risk to allow the users to share everything they want as we have no control over the file contents. However, any restrictions on what to download would also mean a loss of freedom for the user. Furthermore, the necessity to sign all applications (see below) already constitutes quite a harsh system of code security.

With the current implementation of Bluetella, it is not possible to securely authenticate any peer and it is equally impossible to be anonymous. Latter due to the uniqueness of the Bluetooth address, as well as the short physical distance to the next peer. At the moment, the only way in which the integrity of the received data is checked is the file ID (see Section 4.5). Though, because the file ID itself is transmitted without any protection, this is only a partial solution. In addition, all data traffic is transmitted in clear text. Like this, everybody can principally intercept and read it. However, it is not clear who would like to propagate falsified files or secretly eavesdrop files that are shared anyway (especially when you consider that such a person would need to be in immediate physical proximity to the attacked peer).

4.6.1 Signing Applications

MIDP 2.0 brings along a system of trusted and untrusted applications. As this system is described very coherently in [17], the most important facts will be cited from there:

“MIDlet suites can be downloaded from the network in an anonymous fashion, and as such, there are some security and privacy issues that the user may be concerned about: Can a MIDlet read private data and send it to an unknown server? Can it make unauthorized calls that cost money to the user? Can rogue programs run on the device and potentially cause problems?

In MIDP version 2.0, the security model was enhanced to allow MIDlets access APIs that are considered sensitive. For example, making an HTTP connection is one of such sensitive operations because it may involve monetary costs for the user. MIDP 2.0 introduces the concept of trusted and untrusted MIDlets. An untrusted MIDlet suite has limited access to restricted APIs, requiring user approval depending on the security policy of the device. On the other hand, trusted MIDlet suites can acquire some permissions automatically depending on the security policy.

In case an untrusted MIDlet suite tries to invoke a protected API and does not have the permission, for example if the user rejects the permissions prompt, a SecurityException will be thrown. The MIDlet should catch those exceptions and handle them properly.

Permissions are used to protect APIs that are sensitive and
Concepts

require authorization. The MIDP 2.0 implementation has to check whether a MIDlet suite has acquired the necessary permission before invoking the API.

*Protection domains* are a key security concept in MIDP 2.0. A protection domain is a set of permissions and interaction modes. Those permissions can be either automatically granted or deferred until user approval. They are called *allowed* and *user permissions* respectively. When a MIDlet suite is installed, it is assigned to a given protection domain and acquires its permissions and interaction modes.

User permissions may require an explicit approval by the user. The user can either deny the permission or allow it. There are three interaction modes in which user permissions can be granted: *blanket*, *session*, and *oneshot*. When the blanket interaction mode is used, the MIDlet suite acquires the permission as long as the suite is installed, unless explicitly revoked by the user. The session mode requests the user authorization the first time the API is invoked and its validity is guaranteed while any of the MIDlets in the same suite are running. Finally, oneshot permissions request user approval every time the API is invoked. The protection domain determines which of the modes are available for each user permission as well as the default mode.

Each protection domain, except for the untrusted domain, is associated to a set of root certificates. When signing a MIDlet suite, it is necessary to use a public key certificate that can be validated to one of those root certificates. This association will be used to assign the MIDlet suite to a given protection domain. The relationship between root certificates and protection domain is that a domain can be associated to many root certificates, whereas a root certificate is associated to only one domain.

To sign your MIDlet you will need a code-signing certificate conforming to the X.509 Public-Key Infrastructure (PKI) specification. The device will use a set of root certificates to validate the MIDlet suites certificate. Among them it is expected to find the manufacturers root certificate as well as well-known CAs root certificates. Depending on the CAs policy, the certificate can include any number of intermediate certificates that should also be included in the MIDlet.”

This system of certificates probably works pretty fine in the industry. However, it caused us some trouble. While it is possible to create a self-certified, trusted MIDlet for the WTK emulator, this does not work for real phones. The only way to make Bluetella trusted would be to purchase a certificate that is accepted by one of the Certification Authorities supported by Nokia phones. Unless this is done, the user is asked to confirm every single file access done by Bluetella. At VeriSign, such a certificate would
cost around 400 US$ per year.

4.7 User Interface

The possibilities of the MIDP user interface concept are rather restricted. Therefore, it is even more important to design a concise user guidance. In order to ensure to clarity of the handling, the user interface is structured hierarchically. For more information about the user interface, see Appendix C and Section 6.6.

![Screen Shot: Main menu](image)

Figure 4.4: Screen Shot: Main menu
Chapter 5

Bluetella Protocol
Specification

The Bluetella File Sharing protocol (BFS) is inspired by the communication protocol specified by Ganymed Stanek (see [1]). The protocol describes Bluetella messages that are exchanged between mobile phones. The greatest difference in our design is that we completely removed the routing between different devices. For the reasoning described in Section 4.2, we decided to use the idea of user mobility instead of routing. We defined the protocol in such a way that it allows complex search definitions as described in Section 4.1. Additionally, we introduced a new message type (error) for debugging purposes and improved error handling.

5.1 Bluetella Server UUID

UUID stands for Universally Unique Identifier. These 128-bit values are used to unambiguously identify a service running on a Bluetooth device. The UUID for Bluetella is 5e29a2dd10ce41d0bf8be9b7c3e3e5bb (randomly generated).

5.2 Message Flow

Figure 5.1 illustrates the message flow when a client searches for new files. After the connection has been established, one or more search requests are sent (1). In order to distinguish the results later, each search request contains a number that identifies it for this connection (searchIdentifier). The server receives all requests, processes them and replies by sending one or more search results depending on how many files matched the search (2). By setting again the searchIdentifier number, the server can tell the client to which search request the result belongs. The client will then be able to map the result to the corresponding search he sent. If there are no search results, the server sends just one empty search result for this search. Finally, if the client received all results, he will close the connection. In an error case, both, the client as well as the server have the possibility to
Figure 5.1: Messages when searching for files

send an error message (3)(4) before they close the connection indicating the cause. We defined no special action on the reported error codes. The current implementation just logs the cause of the error and then closes the connection. This could be useful for debugging when searching for communication failures.

Figure 5.2 illustrates the message flow when a client wants to download a file from a server. First, the client opens a connection. Then, he sends a file request (1) that specifies the parts of the file he wants to download. The server answers either with a file delivery message that contains the desired file data (2) or an error message (3). The client receives the file delivery header and the appended file data. After the requested data has been transferred, the client closes the connection. The client has the possibility just like the server to send an error message (4) if either the file delivery header is defective or if an error occurs while the file data is being received.
In any case, the sender of an error message closes the connection.

5.3 Message Specification

The protocol consists of five different message types: search request, search response, file request, file delivery and error. All these message types start with the common fields described in Table 5.1. The headerLength defines the length of the protocol header in bytes. Actually, this is the length of the whole packet for all message types except for the file delivery case where the appended file data is considered as the payload. The senderAddress holds the Bluetooth address of the sender. We introduced this field because the Java Bluetooth API provides no means for the server to determine the Bluetooth address of the client that opened a connection. The accepting server thread actually just receives a standard StreamConnection object through its notifier. This yields to the situation that only the client side knows the address of the other side. Because it is useful for the server to know who its clients are, we introduced the senderAddress field. However, it is important to note that there is no way to authenticate this information, i.e. the sender could fill in any address.
### Table 5.1: BFS: Common fields

<table>
<thead>
<tr>
<th>field name</th>
<th>bytes</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>protocolVersion</td>
<td>1</td>
<td>current version = 1</td>
</tr>
<tr>
<td>headerLength</td>
<td>4</td>
<td>BFS header length in bytes</td>
</tr>
<tr>
<td>messageType</td>
<td>1</td>
<td>0 = search request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = search response</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = file request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 = file delivery</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 = error</td>
</tr>
<tr>
<td>senderAddress</td>
<td>6</td>
<td>sender Bluetooth address</td>
</tr>
</tbody>
</table>

5.3.1 Search Request

Additionally to the common fields listed in Table 5.1, a search request consists of the elements described in Table 5.2.

<table>
<thead>
<tr>
<th>field name</th>
<th>bytes</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>searchIdentifier</td>
<td>2</td>
<td>search identifier</td>
</tr>
<tr>
<td>maxNumberOfSearchResults</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>searchObject</td>
<td>headerLength - 15</td>
<td>see Table 5.3</td>
</tr>
</tbody>
</table>

Table 5.2: BFS: Search request

The idea of the `searchIdentifier` is that it uniquely identifies the search to which the search request/response packets belong within the same connection. This makes it possible to return search results for multiple search requests in one single connection. The searching device can restrict the maximum number of returned search results by specifying `maxNumberOfSearchResults`. The `searchObject` (Table 5.3) is a set of fields that mainly define a search name and a list of search attributes. The search name should be some sort of a user readable representation of the actual search. It is used for user interaction and display in the GUI. The search attributes define the actual search. The `searchObject` is quite special as it can be considered as an object that is wrapped in a search request, transferred from peer to peer and ages with every “hop” until its time-to-live has reached zero. Each time the server side receives a search request, the following steps are performed:

1. Extract the `searchObject`.
2. Update the Top10 List (see Section 4.2).
3. Send back search results for files matching the search.

Because the Top10 List will send search requests containing received search objects, we had to make sure that the search information (i.e. the `searchObject`) will not loop around forever. Therefore, a time-to-live field was
5.3 Message Specification

... added to the searchObject definition. Additionally, the sender Bluetooth address of the search is remembered, because the server side must not send back a Top10 search to a host from which the search was received. So, the steps for the server are:

1. Extract the searchObject.
2. Decrement its timeToLive.
3. Update Top10 List if timeToLive is greater than zero and remember the sender address.
4. Send back search results for files matching the search.

<table>
<thead>
<tr>
<th>field name</th>
<th>bytes</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>searchNameLength</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>searchName</td>
<td>searchNameLength</td>
<td>search name</td>
</tr>
<tr>
<td>timeToLive</td>
<td>1</td>
<td>default = 5</td>
</tr>
<tr>
<td>numberOfSearchAttributes</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>searchAttributes</td>
<td>*</td>
<td>see Table 5.4</td>
</tr>
</tbody>
</table>

Table 5.3: SearchObject

As mentioned already, the actual search consists of a set of search-Attributes (Table 5.4). Each search attribute is built upon a file attribute and two logical operators. The file attributes are basically type/value pairs as defined in Table 5.5. The attributeNumber specifies the type of the attribute. See Table 5.14 in Section 5.4 for a complete list of file attribute types. The value itself is the actual encoded data for the attribute. Note that the value field definition of the file attribute depends on the actual type of the value. Currently, there is support for string and integer typed values (Tables 5.6 and 5.7). String comparison matches if the attribute string is a substring of the file information. String matching is not case-sensitive. The logical operators are logicalFunction and isNegated. The logicalFunction specifies the comparison that should be made when the matching is performed. All search attributes are linked with a logical AND. The isNegated tells whether matching for this attribute should be inverted. That means the attribute itself is a logical NOT. A search matches a file if and only if all search attributes match for this file. Here are some examples:

- File Type = “Audio” AND Genre = “Pop” AND Bitrate >= 128 AND NOT Artist = “Britney Spears”
- File Type = “Picture” AND File Name = “Flower” AND File Size <= 1024 KB
### 5.3.2 Search Response

Additionally to the common fields listed in Table 5.1, a search response consists of the elements described in Table 5.8. Again, the search identifier is included. This makes it possible for the receiver to know to which search this result belongs. The `numberOfRemainingResults` indicates how many response packets will follow for this search identifier. Finally, the `fileInfoObject` field describes the matching file with as many attributes as possible. The `fileInfoObject` is shown in Table 5.9. It has three mandatory file attributes:

- file name
- file size
- file extension

If the file has no extension, the corresponding attribute value consists of an empty `String` object. The `fileID` field contains a 16-byte MD5 hash value of the file data.
5.3 Message Specification

<table>
<thead>
<tr>
<th>field name</th>
<th>bytes</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>searchIdentifier</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>numberOfRemainingResults</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>fileInfoObject</td>
<td>headerLength - 15</td>
<td>see Table 5.9</td>
</tr>
</tbody>
</table>

Table 5.8: BFS: Search response

<table>
<thead>
<tr>
<th>field name</th>
<th>bytes</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileID</td>
<td>16</td>
<td>file MD5 hash</td>
</tr>
<tr>
<td>numberOfFileAttributes</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>fileAttributes</td>
<td>*</td>
<td>see Table 5.5</td>
</tr>
</tbody>
</table>

Table 5.9: FileInfoObject

5.3.3 File Request

Additionally to the common fields listed in Table 5.1, a file request consists of the elements described in Table 5.10.

<table>
<thead>
<tr>
<th>field name</th>
<th>bytes</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileID</td>
<td>16</td>
<td>file MD5 hash</td>
</tr>
<tr>
<td>offset</td>
<td>4</td>
<td>file offset in bytes</td>
</tr>
</tbody>
</table>

Table 5.10: BFS: File request

5.3.4 File Delivery

Additionally to the common fields listed in Table 5.1, a file delivery message consists of the elements described in Table 5.11.

<table>
<thead>
<tr>
<th>field name</th>
<th>bytes</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileID</td>
<td>16</td>
<td>file MD5 hash</td>
</tr>
<tr>
<td>payloadLength</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>payload</td>
<td>payloadLength</td>
<td>the actual part of file transmitted</td>
</tr>
</tbody>
</table>

Table 5.11: BFS: File delivery

5.3.5 Error

Additionally to the common fields listed in Table 5.1, an error message (Table 5.12) provides a field containing an numeric error code. A list of defined error codes can be found in Table 5.13.

Error messages can be sent back to inform the other side why a request failed or an operation was interrupted. However, not all of the listed codes
are actually sent back in the current implementation.

If an error 30-35 occurs while receiving a packet, the packet does not adhere to the protocol specification. The reason is given by the error code. Only the server side sends back these error codes for invalid packets. Otherwise the client side could again respond with an invalid packet error too. This could create an endless loop where both sides get stuck in sending error messages to each other.

<table>
<thead>
<tr>
<th>code</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>unspecified error</td>
</tr>
<tr>
<td>2</td>
<td>unexpected packet type</td>
</tr>
<tr>
<td>10</td>
<td>bad search identifier</td>
</tr>
<tr>
<td>20</td>
<td>could not open file</td>
</tr>
<tr>
<td>21</td>
<td>file not found</td>
</tr>
<tr>
<td>22</td>
<td>user canceled transfer</td>
</tr>
<tr>
<td>23</td>
<td>unexpected end of file</td>
</tr>
<tr>
<td>30</td>
<td>invalid packet</td>
</tr>
<tr>
<td>31</td>
<td>unsupported protocol version</td>
</tr>
<tr>
<td>32</td>
<td>bad string representation</td>
</tr>
<tr>
<td>33</td>
<td>unsupported attribute type</td>
</tr>
<tr>
<td>34</td>
<td>unsupported logical search operator</td>
</tr>
<tr>
<td>35</td>
<td>missing attributes</td>
</tr>
</tbody>
</table>

Table 5.13: BFS: Error Codes
### 5.4 File Attributes

<table>
<thead>
<tr>
<th>file attribute</th>
<th>examples</th>
<th>attribute number</th>
</tr>
</thead>
<tbody>
<tr>
<td>search in all attributes</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>file name</td>
<td>mycat.jpg</td>
<td>2</td>
</tr>
<tr>
<td>file type</td>
<td>audio, picture, movie</td>
<td>3</td>
</tr>
<tr>
<td>file extension</td>
<td>mp3, jpg, bmp, mpg, txt</td>
<td>4</td>
</tr>
<tr>
<td>fileID</td>
<td>[16 Byte MD5 Hash]</td>
<td>5</td>
</tr>
<tr>
<td>file size (bytes)</td>
<td>3300</td>
<td>6</td>
</tr>
<tr>
<td>artist</td>
<td>U2</td>
<td>32</td>
</tr>
<tr>
<td>song title</td>
<td>Bloody Sunday</td>
<td>33</td>
</tr>
<tr>
<td>album</td>
<td>The Joshua Tree</td>
<td>34</td>
</tr>
<tr>
<td>genre</td>
<td>Rock, Pop, Ska</td>
<td>35</td>
</tr>
<tr>
<td>bit rate (kbit)</td>
<td>128</td>
<td>36</td>
</tr>
<tr>
<td>author</td>
<td>Umberto Eco</td>
<td>64</td>
</tr>
</tbody>
</table>

Table 5.14: File Attributes

<table>
<thead>
<tr>
<th>file type</th>
<th>corresponding file extensions</th>
<th>encoded number</th>
</tr>
</thead>
<tbody>
<tr>
<td>everything</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>audio</td>
<td>mp3, ogg, wma, wav</td>
<td>2</td>
</tr>
<tr>
<td>ringtones</td>
<td>mid, midi</td>
<td>3</td>
</tr>
<tr>
<td>text files</td>
<td>txt, doc, rtf</td>
<td>4</td>
</tr>
<tr>
<td>videos</td>
<td>mpg, mpeg, avi, wmv</td>
<td>5</td>
</tr>
<tr>
<td>pictures</td>
<td>jpg, jpeg, png, bmp, gif</td>
<td>6</td>
</tr>
<tr>
<td>applications</td>
<td>exe, jad</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 5.15: File Types
Chapter 6

Implementation

6.1 Class Diagram

Figure 6.1 depicts the most important classes and their role in the Bluetella application. All these classes will be described in the following sections.

![Bluetella class diagram](image)

Figure 6.1: Bluetella class diagram

6.2 Communication

In order to begin to communicate, one has first to know what devices (i.e. Bluetooth addresses) are in range and which services they provide. For this discovery phase, we implemented a server and a client process which
are each running in their own thread. Once other peers have been found, a connection can be established. As explained in Section 2.1, either L2CAP or RFCOMM can be used. Bluetella uses only RFCOMM connections through the Serial Port Profile (SPP). SPP is a Bluetooth profile that provides a stream-based interface to the RFCOMM protocol. In Java, this is implemented by JSR-82. Once the connection has been opened, the normal Java stream classes such as InputStream and OutputStream can be utilized for data transfer.

In order to establish an SPP connection to a device, the connection URL has to be known. An example connection URL looks like this:

```
btspp://1234567890ab:1
```

This address states that the client should use SPP to establish a connection to the service that is identified with server channel 1 on the device whose address is 1234567890ab. The server channel identifier must be assigned by the server when the service is registered to the server’s Service Discovery Database (SDDB). See [12] for more details.

6.2.1 Discovery

Server Process

The task of the ServerProcess is to listen for connection requests. For each new connection, a thread called ServerRequestHandler is spawned that answers incoming requests. In the following, the most important steps of this procedure are explained in more detail with references to actual code snippets.

The first step is to make sure that the device can be seen by others. The two lines

```java
LocalDevice local = LocalDevice.getLocalDevice();
local.setDiscoverable(DiscoveryAgent.GIAC);
```

set the device to the General/Unlimited Inquiry Access Code (GIAC). This allows other devices to discover it. In order to be able to listen for connections, an URL has to be specified that identifies the service the server provides. For Bluetella, this so-called service URL is defined as:

```
String url = "btspp://localhost:5e29a2dd10ce41d0bf8be9b7c3e3e5bb;name=Bluetella"
```

btspp://localhost means a Serial Port Profile service on the local device. Followed by the 32-digit hexadecimal Universal Unique Identifier (UUID) that identifies the actual service. The UUID for Bluetella has been defined in the protocol specification (Chap. 5). The attribute name=Bluetella is optional and only appended for human readability.

Now, in order to actually listen for incoming connections on this service, a StreamConnectionNotifier is created:
6.2 Communication

StreamConnectionNotifier notifier;
notifier = (StreamConnectionNotifier) Connector.open(url);
StreamConnection conn = notifier.acceptAndOpen();

The method notifier.acceptAndOpen() blocks until a client wants to open a connection. The return value is a reference to a StreamConnection object from which normal input and output streams can be opened:

InputStream input = conn.openInputStream();
OutputStream output = conn.openOutputStream();

By putting notifier.acceptAndOpen() in a while loop, the server process receives a StreamConnection for every client that connects. For each new connection, a ServerRequestHandler thread is started that opens the streams and answers incoming requests.

Client Process

The ClientProcess searches for devices running Bluetella. For each found Bluetella server, a SearchProcess thread is started that automatically searches results for all entries in the search lists (see Section 6.5.2).

In order to search for services (in our case: the Bluetella service), we need to know the devices in range first. JSR-82 provides a DiscoveryAgent class and a DiscoveryListener interface for device and service search. The lines

LocalDevice local = LocalDevice.getLocalDevice();
DiscoveryAgent agent = local.getDiscoveryAgent();

create a new DiscoveryAgent. To start a search for other devices, the method call

agent.startInquiry(DiscoveryAgent.GIAC, clientProcess);

will set the device to inquiry mode and detect all discoverable neighbours (devices in GIAC mode). The second parameter clientProcess specifies the DiscoveryListener that handles the following events: Whenever a new device has been found, the system calls the method deviceDiscovered() with an object representing the remote device as first argument. When the device has finished the inquiry phase, the method inquiryCompleted() will be called indicating whether the search was successful or an error has occurred.

Now, hopefully having a list of devices, they can be searched for appropriate services. For each remote device btDev a service search is started with:

agent.searchServices(attrSet, uuidSet, btDev, clientProcess);

The arguments attrSet and uuidSet indicate the type of the searched service, with the uuidSet containing the Bluetella UUID. The last argument
clientProcess again specifies the DiscoveryListener used for the service search. This time, the DiscoveryListener handles the following events: Whenever a matching service (with respect to attrSet and uuidSet) has been detected, the method servicesDiscovered() is called and we receive a ServiceRecord. From this ServiceRecord, we can extract a connection URL used for connecting to the host that offers the service (see below). The connection URL should not be mixed up with the service URL used for advertising a new service. When the service search finished, the system invokes serviceSearchCompleted().

ServiceRecord record;
...
String url = record.getConnectionURL(...);
...
StreamConnection conn;
conn = (StreamConnection) Connector.open(url);

In our implementation of the ClientProcess, the device and service search is performed by the ClientProcessTask. The ClientProcessTask is implemented as a TimerTask that is called in regular time intervals. At first, it performs a device search run. If this has been successful, it starts a service search on each device sequentially (some mobile phones do not support simultaneous service searches – the Nokia 6630 is an example for this). Finally, a thread called SearchProcess is started for each new Bluetella server device. This thread searches for files on this specific host.

6.2.2 Information Transfer

Server Request Handler

The ServerRequestHandler is started for each incoming connection by the ServerProcess (see Section 6.2.1). After opening input and output streams on the connection, the ServerRequestHandler waits for data in its own thread. For every received request (search request or file request), it will spawn a new thread called RequestProcessor that processes the request. The RequestProcessor threads will send back search response or file delivery messages to the requesting client through a synchronized method called ServerRequestHandler.sendPacket(). This makes sure that the different packets do not get mixed up when they all are sent through the one output stream of the connection. If the RequestProcessor was started on a search request, it will insert the contained SearchObject into the search cache and update the Top10 List if the time-to-live value permits it (cf. Section 5.3.1). If an error occurs while receiving or sending packets, the ServerRequestHandler terminates itself, including all RequestProcessor threads it has started and deregisters itself with the ServerProcess.
6.2 Communication

Figure 6.2: Search Process

Search Process

The SearchProcess is a separate thread that is started by the Client-Process for each Bluetella peer found (see Figure 6.2). The following steps are performed in one search run:

1. Open the connection.

2. Process the search lists:

   (a) Choose the next searches from user and/or Top10 search list. By default, at most five searches are packed together. Searches from the Top10 List are only considered if the Top10 search mode has been activated.
(b) Send this sequence of searches.
(c) Wait for search response packets for those searches. As soon as a search response packet arrives, the file information is extracted and inserted into the corresponding result list.
(d) When all search response messages have been received, go to 2a until all entries in the considered search lists have been searched once.

3. Close the connection.

The **SearchProcess** will repeat the search run after a certain time interval, since it is possible that the number of available files on the remote device have changed in the meantime. If an error occurred while performing a search run, the search process terminates and deregisters itself with the **ClientProcess**.

**Download Process**

![Diagram of Download Process]

Figure 6.3: Download Process

The **DownloadProcess** is an independent thread that monitors the download list (see Fig. 6.3). If there is a file in the list that should be downloaded,
the DownloadProcess will start a FileTransfer thread for the file, taking into account the maximal number of allowed simultaneous downloads. The FileTransfer opens a connection in a separate thread and tries to retrieve the desired file. If it fails to download on one URL, it will try it on the next URL specified by the DownloadObject. The FileTransfer terminates if either the file has been successfully downloaded or it failed on all URLs to retrieve the file. One thing to note is that all entries in the download list are marked with a time stamp that contains the last time a FileTransfer tried to download it. This prevents the DownloadProcess from repeatedly spawning file transfers for a file that cannot be downloaded at the moment.

As stated before in Section 4.3, the files are downloaded as simple byte streams. A start offset is specified to resume broken downloads. Unfinished files are being stored in the Bluetella/Incoming/ directory. The integrity of downloaded files is checked by calculating the file ID of the new data and comparing it to the entry from the download list. When the file has been successfully downloaded, it is moved to the Bluetella/Downloaded/ folder. The user can select the directory Bluetella/ in the preferences dialog of the user interface.

If the FileTransfer receives an error message with code 21 (file not found), it removes the processed URL from the result and the download list. This causes Bluetella to stop requesting the file from this server as it is not available anymore.

### 6.3 Packets

This section focuses on the actual sending and receiving of packets (messages). All five message types (confer protocol specification in Chapter 5) are represented in the Packet class. The Packet class provides a special constructor for every message type taking the message data as arguments.

```java
/** Create an empty packet */
public Packet()

/** Create a search request packet */
public Packet( int searchIdentifier, int maxResults, SearchObject searchObject )

/** Create a search response packet */
public Packet( int searchIdentifier, int numberOfRemainingResults, FileInfoObject fileInfoObject )

/** Create a file request packet */
public Packet( byte[] fileID, int fileOffset )

/** Create a file delivery packet */
public Packet( byte[] fileID, int payloadLength, InputStream fileIn )
```
/** Create a new error packet */
public Packet(int errorCode)

With Packet.send() the message is sent to the specified output stream. In order to receive a message, an empty packet is created first. Packet.receive() will then receive a message from the specified input stream. All data fields from a received packet can be read out via get-methods.

Sending these messages over a byte stream means that the SearchObjects or the FileInfoObjects have to be first converted into byte arrays. All objects that are transmitted (or stored in the application’s record store) implement therefore the two methods

Serializable.toStream(OutputStream out)
Serializable.fromStream(InputStream in)

of the Serializable interface we defined. All objects that implement this interface can be read from an input stream or written to an output stream according to the Bluetella protocol specification. For instance, SearchObject.toStream() is called to write the SearchObject to the output stream in Packet.send() when sending a search request.

There is a serious issue with Nokia’s implementation of the InputStream class when using in connection with the Bluetooth Serial Port Profile: As stated in [26], the InputStream cannot cope with arrays larger than 512 bytes when reading from a stream. Using larger arrays will cause stability problems. We tried to circumvent this problem by making the Packet class reading and writing in chunks of at most 512 bytes.

6.4 Indexing Files

Bluetella has to know about the files available on the user’s mobile phone. The FileIndexer class provides a method to automatically scan files in the folders which the user selected to be shared. The FileIndexer knows actually two running modes. One is used to scan all shared files in a background thread. The second mode is a method (indexSingleFile()) for indexing a single file. This method is called by the FileTransfer when a new file has been downloaded.

Ideally, all possible meta information of a file is extracted (e.g. ID3 tags in audio files). However, for simplicity, in our implementation only the following file information is assembled:

• file name
• file size
• file extension
• file type (based on file extension)
• file ID
6.5 List Data Structures

- last modified

The last modified time is used to know whether the file needs rescanning. The file ID is a 16-byte MD5 hash. All this information is collected in a MyFileObject (see Section 6.5) and inserted into the My Files List.

6.4.1 File ID

We specified the file ID to be a cryptographic hash of the file content. If it comes to file sharing, most applications use SHA-1 (Secure Hash Algorithm) as it provides better security compared to the MD5 algorithm. We decided to use MD5 as we thought it well enough for our purpose and because it is faster to compute than SHA-1. For mobile phones, this is thought to be quite an important consideration. For the MD5 computation, we use the library provided by the Bouncy Castle Cryptography Package [25] for J2ME. This library allows to compute SHA-1 too, so it would be very easy to switch the hash algorithm in a later Bluetella version, if desired.

6.5 List Data Structures

This section describes the structure and the implementation of the different lists we use to store file and search data. All the lists below are managed by the Lists class:

Vector myFilesList;
Vector searchList;
Vector top10List;
Vector searchCache;
Vector resultListsList;
Vector top10ResultListsList;
Vector downloadList;
Vector uploadList;

They are saved upon program termination in the record store of the application and restored when Bluetella starts again (see methods Lists.save() and Lists.load()).

For the definitions of FileInfoObject, FileAttributes, SearchObject and SearchAttribute, please refer to Chapter 5.

6.5.1 My Files List

The My Files List holds information about all the shared files on the local mobile phone. Figure 6.4 shows the structure of this list. The myFilesList is a Vector that holds MyFileInfoObjects which contain the path of the file and a FileInfoObject describing the file by providing a set of FileAttributes.
6.5.2 Search Lists

The Search List contains the user defined searches. One search is represented as a SearchObject. The Top10 List contains the ten most wanted searches (confer Section 4.2). The entries from both lists are searched by the SearchProcess described before.

In order to determine the most wanted searches, a cache is needed that stores a certain amount of received searches. The searchCache list is used for that. The search cache is kept up to date by the method

Lists.addSearchToSearchCache()
which is called by the `RequestProcessor` threads while processing search requests. This method manages the `searchCache` such that the elements are sorted in descending order by their search frequency. The `top10List` thus contains actually just the first ten entries of the `searchCache`.

Figures 6.5 and 6.6 show what is stored in the two search lists and the search cache. The `SearchCacheObject` is an extended `SearchObject` that additionally contains a counter, a timestamp and a list of URLs that sent this search. The timestamp and the URLs are used to prohibit endless loops of `SearchObjects` between devices. See Sections 4.2.2 and 5.3.1 for a discussion of this problem.

### 6.5.3 Result Lists

There are two result lists. One contains the results for user searches, the other for the automatic most wanted searches. The structure of a result list is shown in Figure 6.7. A result list contains a list of `ResultListObjects`.

![Figure 6.7: List of Results](image)

The `ResultListObject` contains additionally to the search name a list of `ResultObjects`. That is why a result list is actually a result list list. The `ResultObjects` contains the actual file information (`FileInfoObject`) and a set of connection URLs where the file can be downloaded.

Both result lists are kept up to date by the `SearchProcess` threads. Whenever they receive a search response, the method

```
Lists.addResultToResultList()
```

is called.

### 6.5.4 Download List

All files that the user or the automatic Top10 search selects for download are stored in the `Download List`. Figure 6.8 shows the structure. The
downloadList is a list of DownloadObjects that contain – besides the corresponding FileInfoObject – a set of URLs where the file can be downloaded. When a file has been successfully downloaded, it will be deleted from this list.

![Download List structure](image)

**Figure 6.8: Download List structure**

### 6.5.5 Upload List

The Upload List is a list of currently active file uploads. In our implementation, this corresponds to the number of running RequestProcessor threads that perform a file delivery. Figure 6.9 shows the information that is stored in the uploadList.

![Upload List structure](image)

**Figure 6.9: Upload List structure**

### 6.6 User Interface

Bluetella’s user interface is implemented in the class UserInterface. It uses the LCD User Interface API of MIDP 2.0. All screens, commands and other graphical features of Bluetella are handled by this class. You can find a chart of the Bluetella User Interface in Appendix C.

There are three main classes of objects that are used in the user interface:

- **Forms**: This object represents a screen. It can be filled with a variety of different Items, among them text fields or choice groups.

- **Lists**: This object represents a screen as well. For each element of the list, a line is shown on the screen. Lists provide more functionality
to manage the objects in the list than Forms. However, there is no possibility to display anything else than strings.

- **Commands**: This objects can be attached either to a List or to a Form. Each Command generates an entry in the menu of the List or Form it was attached to.

In Bluetella, almost all of these objects are final elements of the User-Interface class. The respective commands and the list entries are added in the constructor.

How the Forms, Lists and Commands actually are displayed is not defined by MIDP and differs from device to device. In fact, there even are some minor differences between the emulator and the phone.

![Figure 6.10: Screen shot: Download information](image)

6.6.1 Command Handling

Every time the user selects a menu item, the corresponding Command is handed over to the commandAction method of the UserInterface that implements the CommandListener interface. In this method, there is a big decision tree to find out what action has to be taken. First, the tree branches off for the screen and secondly for the different Commands of this screen. There, the appropriate function for this menu item is called.
Chapter 7

Test Results

In order to test the possibilities and limitations of Bluetella on Nokia 6630 mobile phones, a number of tests was conducted. The firmware version of the used devices was V.2.39.15 (SymbianOS).

7.1 Functional Tests

7.1.1 Discovering Peers

The discovering process is the most sensitive part of Bluetella. The device and service search functionality (described in Section 6.2.1) does not always produce accurate results with the phones we used, causing Bluetella not to find all peers in range at times. One problem are the limitations for simultaneous connection setup in a piconet (see discussion in Appendix D.1): As service discovery involves opening a connection, devices can disturb each other during the discovery process.

7.1.2 Correctness of the Results

In order to test whether the RequestProcessor (described in Section 6.2.2) returns correct results, we simply designed a great number of advanced searches and compared the results to the results we expected. We did not discover any mistakes.

7.1.3 Most Wanted List

As we only had two devices at our disposal, we were not able to conduct extensive examinations of the Most Wanted functionality (see also Future Work Section 8.2.3). However, we could see that the Most Wanted List was updated correctly.

7.1.4 Downloading

Principally, downloading does work. However, we did encounter a problem while transferring files with sizes greater than about one hundred kilobytes:
With a non-reproducible pattern, the uploading phone performed a complete reboot. We could not exactly locate the cause of this bug, but we can speculate that this behavior is caused by a bug in Nokia’s firmware (cf. Appendix D.2)

7.1.5 More than One Download

We also tested what happens if we try to download several files at the same time. The result was that one file was downloaded after the other. Unfortunately, we did not have the possibility to test if everything works fine when two files are downloaded from two different peers. However, we tested if two devices can download files from each other at the same time. This actually was possible, but one the downloads were active only in an alternating manner (further explained in Appendix D.1). The ability of a device to download switched every 120-125 seconds. We assume that 120 seconds is the time after which the roles within the piconet are assigned anew.

7.1.6 Interrupted Downloads

We tried out whether it was possible to resume an interrupted download. This did work just as well when the application was terminated by the user as when the connection was broken because one device left the radio range of the other. Furthermore, stopping uploads or downloads from the user interface worked without problems.

7.1.7 User Interface

To test the comprehensibility of the user interface, we elected two test persons and gave them some tasks to solve. They did not have any knowledge about Bluetella beforehand, however, they were familiar with other file sharing applications. One test person uttered some criticism on minor graphical details (which to solve was beyond our means), but both were able to share and download files within a few minutes.

7.1.8 Energy Consumption

With one recharge of the battery\textsuperscript{1}, it was possible to run Bluetella for more than five hours. During this time Bluetella was constantly exchanging data. We think that this is quite a nice value and should not annoy the user too much.

\textsuperscript{1}The Nokia 6630 has a 900 mAh Li-Ion battery
7.2 Performance Tests

7.2.1 Discovering Peers

In order to measure the time until a connection to a peer is established, we proceeded in the following way: Bluetella was running on one device. Then, Bluetella was started on the other device as well. The timer was started at the moment the Bluetooth connection was granted for Bluetella and stopped when the first of the two mobile phones discovered. The device discovery rate was set at 30 seconds. There were 4 or 5 other Bluetooth devices in the room.

The results were as follows: It took 20 to 40 seconds to establish the first connection with an average of around 29 seconds.

7.2.2 Creating the MD5 Hash

The computation of MD5 hashes is surprisingly fast: The hash calculation of a 4.6 MB file takes 31 seconds or 6.7 seconds per megabyte. This is substantially faster than in the emulator.

7.2.3 Download Speed

In theory, Bluetooth provides a bandwidth up to 721 kbit/s on an asymmetric channel (see Section 2.1.1). In practice, the throughput is considerably less once things like overhead and error control are factored in. Additionally, the implementation of the Serial Port Profile (SPP) is not yet very mature on Nokia mobile phones (cf. [26]).

We conducted a series of tests to measure the throughput reached for transferring files. We made experiments with files of different size and in environments with different number of other Bluetooth devices in radio range. The measured download speeds lay between 39.2 kbit/sec and 70.2 kbit/sec. This is substantially less than the theoretically achievable values. We can only speculated where this comes from. One possible explanation could be that SPP utilizes the slower reverse channel of the ACL link (see Section 2.1.1). However, we are not able to examine this hypothesis further, as we only can use the functionality provided by J2ME.

Besides, we observed a tendency that bigger files were downloaded quicker than smaller ones. This lets us assume that there is a considerable overhead until a connection is established.

7.2.4 Memory Usage

In the tests, Bluetella allocated between 700 and 1500 KB memory, depending on the number of connections and the size of the lists.
Chapter 8

Conclusion

We made the experience of developing at the forefront of an evolving technology. Some of the standard APIs we used were issued a quarter of a year before we started our thesis. In an almost monthly rhythm, new and more powerful tools were published. On one hand, it is exiting to see things progress that quickly and to try to do one’s work in such a way that it is compatible with future developments. On the other hand, it is also a challenge to work with technologies that are not a hundred per cent mature yet.

8.1 Summary

In the course of this thesis, the Bluetooth API for J2ME enabled phones was studied. The properties of different phones and development platforms were examined and Nokia’s model 6630 was chosen to test the Bluetella application. A concept was sketched how a file sharing application for Bluetooth phones was designed the best. Instead of a multi-hop routing protocol, an approach was chosen that bases on the exploitation of the natural mobility of the user and on the selective replication of frequently searched files. Consequently, this concept was implemented and tested.

The test showed that Bluetella does work as projected. Nevertheless, there are some drawbacks: The connection establishment between two Bluetella peers often takes a tiresome time. The used Bluetooth devices were not perfectly stable and produced crashes in some cases. What is more, lacking support for scatternets restricts the number of simultaneous connections. Last but not least, the user mobility concept would need some further investigation. Over all, Bluetella does serve the intended purpose, but of course, it does not even come near to the convenience and the speed of Internet file sharing applications.
8.2 Future Work

8.2.1 Signing the Application

If our work is pursued in any kind of way, we strongly recommend to sign the application. The constant security warnings are so annoying that it surely is worth the investment in buying a code signing certificate (see Section 4.6.1).

8.2.2 Additional Features

There are a few features that could be introduced for convenience: The most important of them would be to enhance the file indexing so that it can extract meta information from media files such as ID3 tags from MP3 files.

8.2.3 Most Wanted List

Some improvements were proposed for the Most Wanted List:

- One idea was to weight the entries of the Most Wanted List not only by the number of times a search was received, but by the “grade of acquaintance”\textsuperscript{1} of the peers that issue a search.

- Another idea was to download not only the most wanted files from the encountered peers, but all their files. This would enlarge the amount of replicated data drastically.\textsuperscript{2}

There would be three ways to test how these changes would affect the performance of the file sharing network:

1. After making some assumption about the behavior and especially the mobility of the users, a corresponding theoretical model of the network could be created. Using this model, the consequences of various changes to the Most Wanted List could be estimated.

2. An emulator could be utilized to simulated the response of the Most Wanted System to various modifications. However, the current emulators support a maximum of 8 devices to be running at the same time and they do not support any modeling of mobility, nor of the time delay at the device search and the connection establishment. Furthermore, they assume that there is support for scatternets, which is not the case with real devices.

\textsuperscript{1}The “grade of acquaintance” would be determined by how many times a certain peer had been seen before and thus, how big the probability is that that peer will be met again.

\textsuperscript{2}For public file sharing of music files, this certainly would not be a good option. As only a small fraction of all such files can be stored on one device, the memory of the phones would soon be filled with more or less arbitrary files. However, for a setting with only a limited number of shared files, this could be a reasonable approach.
8.2 Future Work

As the emulators communicate via the TCP/IP stack of the used computer, it may be possible to simulate some of the mentioned effects by dynamically opening and closing the ports used by the emulators. However, this idea has never been tried out in practice yet.

3. The most promising, but also most expensive and time-consuming approach would be to test the different proposals with real phones in practice.

8.2.4 Security

Denial of Service (DoS)

One open problem with the current implementation are possible DoS attacks. One was already mentioned in Section 4.2.2. In order to prohibit or at least reduce the impact of such attacks, one should introduce limits for:

- Number of files downloaded for an Top10 List entry.
- Amount of space reserved for Most Wanted file cache.

Authentication

Theoretically, it would be possible to authenticate all peers securely by using some kind of public key infrastructure. Though, we do not consider this to be worth the loss in convenience, as explained in Section 4.6.

8.2.5 Other Devices

It would also be interesting to test Bluetella on other devices than Nokia 6630 and investigate if the present results will be the same. However, at the time of the writing of this thesis, the only devices that supported all necessary standards were from Nokia as well. So, no great differences in the results would be expected.

8.2.6 From RFCOMM to L2CAP

In order to improve the performance and stability of Bluetella, especially on the currently used Nokia phones, it would be advantageous to use L2CAP instead of RFCOMM (see Nokia’s Known Issues document [26]). However, this would require to design and implement a transport layer, that is able to guarantee the reliability and the order of transmitted L2CAP packets. Alternatively, one could wait for firmware upgrades that improve and stabilize the implementation of the Serial Port Profile.

8.2.7 Is Bluetooth Really the Way to Go?

Generally, it is to be considered if Bluetooth really is the best option to realize a wireless file sharing network. The limitations we found, when it comes to connection establishment or search range, are considerable. With
the quickly progressing development of wireless technologies, it is conceivable that mobile devices will provide other and better means of mobile connectivity soon, most notably WLAN\(^3\).

\(^3\)We also took a look at ZigBee (a new technology based on the IEEE 802.15.4 standard, see [27]) as a potential alternative to Bluetooth. However, as ZigBee’s bandwidth is even more restricted than the bandwidth of Bluetooth and since there are no ZigBee devices available at the moment, we would not recommend to investigate this option any further for Bluetella.
Appendix A

Bluetella User Manual

A.1 Installation

The installation of Bluetella depends on the phone you use. We will only explain the procedure for Nokia phones here.

Together with the mobile phone, Nokia provides a variety of software, the so called *Nokia PC Suite*. Among other useful tools, it contains an application installer. In order to install Bluetella, start the Nokia Application Installer and connect the phone using the USB wire. On the left side of the screen (see Figure A.1), you need to browse to the installation file *Bluetella.jar*, which is delivered on the Bluetella CD or which can be generated in an IDE (e.g. Netbeans). Press the green arrow to install the application. Now, the installation process continues on the mobile phone.

You will be asked if you want to continue although the application is...
untrusted. Select Yes. You might also be asked if you want to replace an older version. Select Yes again. Now you can specify, where to store the application. If there is an older Bluetella version on your phone, the next dialogue will ask whether you want to save existing application data. Select No, if you are unsure.

When you start Bluetella for the first time, you will be prompted an error message and later, you will be asked to choose a folder in which Bluetella will save downloaded files. Please select a directory and then press Back once to get to the main screen. Now, you are member of the highly exclusive Bluetella community and can start sharing and searching files.

A.2 Menu Navigation

The main menu of Bluetella contains six entries:

- The first entry leads to the search menu. The figure next to the caption indicates the number for searches that are active.

- The second menu starts the result menu. The figure indicates the number of searches that produced a result.

- The third menu leads to the transfer menu. The figures next to the caption indicate how many downloads and how many uploads are currently active.

- The fourth menu entry brings you to the information menu. The figure next to the caption indicates the number of peers that are connected the the phone at the moment.

- The fifth entry starts the preferences menu.

- The last entry ends Bluetella.

A.2.1 Search Menu

Enter a simple search (see Section 4.1.2). The search will be started, as soon as you select Add Search.

<table>
<thead>
<tr>
<th>Manage Searches</th>
<th>View or remove searches.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advanced Search</td>
<td>Create more complex searches. Add a new condition by choosing Add Criteria. The search will be started as soon as you select Add Search.</td>
</tr>
</tbody>
</table>

A.2.2 Results Menu

The results are listed by their corresponding search name. The figure next to the caption indicates the number of results found for this search. Inspect a search name by selecting the corresponding line.
A.2 Menu Navigation

**Results for a Search**
View and remove results of a search. Select a file for download. (The figure next to the caption indicates the number of peers that offer this file)

<table>
<thead>
<tr>
<th>A.2.3 Transfer Menu</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Download</strong></td>
</tr>
<tr>
<td><strong>Upload</strong></td>
</tr>
<tr>
<td><strong>Downloaded Files</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A.2.4 Info Menu</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Most Wanted</strong></td>
</tr>
<tr>
<td><strong>Devices in Range</strong></td>
</tr>
<tr>
<td><strong>Shared Files</strong></td>
</tr>
<tr>
<td><strong>Show Log</strong></td>
</tr>
<tr>
<td><strong>Memory Usage</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A.2.5 Preferences Menu</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Set Bluetella Folder</strong></td>
</tr>
<tr>
<td><strong>Show Shared Folders</strong></td>
</tr>
<tr>
<td><strong>Add Shared Folder</strong></td>
</tr>
<tr>
<td><strong>Scan for New Files</strong></td>
</tr>
<tr>
<td><strong>Edit Settings</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Appendix B

CD Contents

The included CD-ROM contains the following directories:

/bin
Binary version of Bluetella, ready to install on a mobile phone.

/doc/report
Final report (with \LaTeX source).

/doc/presentation
/doc/zwischenbericht
The final presentation as well as the “Zwischenbericht”.

/doc/bluetella-javadoc
Javadoc, generated from Bluetella’s source code.

/doc/misc
Further documents, specifications.

/src
The Bluetella source code.
Appendix C

User Interface

Figure C.1: Bluetella User Interface
Appendix D

Known Issues

D.1 Connectivity Limitations

In Section 2.1.2, the difference between scatternet and piconet was explained, and that a Bluetooth device can only be master of one piconet at a time. The device that initiates the connection is also the master of the piconet (per default). It was also mentioned that the Nokia 6630 does not support scatternets\(^1\).

\[ \text{Figure D.1: Piconet and its implications on connection establishment. On the left, there are two devices that have established a connection (1).} \]

This has some implications that will be explained here. Figure D.1 shows three Bluetooth devices that are in range of each other. On the left side, two devices are in the same piconet and the master (M) has initiated a connection (1) with the slave (S) already. There are some limitations that

\(^1\)Actually almost none of the currently available Bluetooth devices support scatternet.
have an impact on other connections at the same time if no support for
catternet is given. For instance, it is not possible for the slave to establish
a connection (3) to other devices now because this would result in changing
the piconet and, thereby, interrupt the ongoing connection (1). However,
the master (M) could establish a second connection (2) to the third device
without problems. Also, other piconets in the same range between other
independent devices do not interfere with this one.

By design, each Bluetella device is server and client at the same time.
Thus, every device will try to establish connections simultaneously to all
other devices in range. (One thing to note is that being a server results in
being the slave in a piconet because the client starts the connection, and not
vice versa.) It is clear that this leads to some problems with the piconets,
i.e. the devices cannot establish all their connections at the same time.

In our tests with two devices, this resulted in the situation that only
one Bluetella device was in an active connection at a time. Nevertheless,
we were able to search and download files with Bluetella on both sides. One
reason is that most connections are opened in order to perform a search for
files which does not take a very long time usually. Secondly, Bluetooth (or
the used phone) seems to handle colliding connections in quite a fair way.
Longer file transfers were interrupted sometimes letting the other device
open a connection (see also Section 7.1.5). A short test with four devices
showed similar effects.

D.2 Stability

There is a bug when downloading larger files. This problem was already
mentioned in Section 7.1.4. Sometimes, the uploading device instantly re-
boots. We could not track down the problem and don’t know why this
happens. To the best of our knowledge, the bug is caused somewhere in
Nokia’s firmware. We used version V.2.39.15 (SymbianOS).

D.3 Shutdown Duration

Access to the application’s record store is rather slow. This is especially
noticeable when the user wants to terminate Bluetella because then, all
internal lists are written to different record stores. Even with just a few list
entries this can take up to 15 seconds.
Appendix E

Assignment

Winter 2004/2005

Semester Thesis
for
Andreas Weibel and Lukas Winterhalter

Supervisor: Matthias Bossardt
Co-Supervisors: Vincent Lenders and Martin May

Issue Date: 18.10.2004
Submission Date: 7.2.2005

E.1 Introduction

Mobile phones of the latest generation feature a Java virtual machine (J2ME) and a Bluetooth stack for short range communications. In this project, a file sharing application must be implemented in Java. This application allows a user to discover and exchange files (e.g. ring tones, background pics, etc.) within communication range.

In [1] a file sharing application was demonstrated on mobile phone emulators that provided file sharing across multiple hops. At that time, however, available hardware did not allow the application to be implemented on real phones. Depending on the capabilities of currently available hardware, new Bluetella variants may use single or multi hop communication.

This work is part of the Blue* project [2], which aims to develop networking applications for Bluetooth-based ad-hoc networks.
E.2 Assignment

E.2.1 Objectives

The goal of this project is to implement a working prototype of the Bluetella application. The design of the application should be such that its parts can be reused for other Bluetooth enabled applications.

E.2.2 Tasks

- Get familiar with the JVM and related APIs provided by the latest models of mobile phones. In particular study the Bluetooth-API.
- Get familiar with the former semester thesis on Bluetella [1].
- Identify mobile phones with Bluetooth support which are suitable for the Bluetella application. Categorize different models based on a set of chosen characteristics. This task should be done in collaboration with the participants of the “BlueDating” thesis.
- Propose different technical approaches for the file sharing application and compare them with each other. Select the most promising alternative.
- Design and implement the protocols for the chosen alternative.
- Define and set up a demonstration scenario of the Bluetella. Preferably the demonstration should run on currently available mobile phones.
- Document your work in a detailed and comprehensive way. We suggest you to continually update your documentation. New concepts and investigated variants must be described. Decisions for a particular variant must be justified.

E.3 Deliverables and Organisation

- Generally students and advisor meet on a weekly basis to discuss progress of work and next steps. If problems/questions arise that cannot be solved independently, the students may contact the advisor anytime.
- At the end of the third week, a detailed time schedule of the semester thesis must be given and discussed with the advisor.
- At half time of the semester thesis, a short discussion of 15 minutes with the professor and the advisor will take place. The student has to talk about the major aspects of the ongoing work. At this point, the student should already have a preliminary version of the table of contents of the final report. This preliminary version should be brought along to the short discussion.
At the end of the semester thesis, a presentation of 15 minutes must be given during the TIK or the communication systems group meeting. It should give an overview as well as the most important details of the work. Furthermore, it should include a small demo of the project.

The final report may be written in English or German. It must contain a summary written in either English or German, the assignment and the time schedule. Its structure should include an introduction, an analysis of related work, and a complete documentation of all used software tools. Related work must be referenced correctly. See http://www.tik.ee.ethz.ch/flury/tips.html for more tips. Three copies of the final report must be delivered to TIK.

Documentation and software must be delivered on a CDROM.
Assignment
Appendix F

Thesis Timetable
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1st Nov.</td>
<td>15 Nov.</td>
</tr>
<tr>
<td>22 Nov.</td>
<td>29 Nov.</td>
</tr>
<tr>
<td>3 Dec.</td>
<td>10 Dec.</td>
</tr>
<tr>
<td>31 Dec.</td>
<td></td>
</tr>
</tbody>
</table>

**Figure F.1: Bluetella Thesis Timetable**
Bibliography


   http://java.sun.com/j2me/j2me-ds.pdf

[12] Sun Microsystems: Java™ APIs for Bluetooth™ Wireless Technology
   (JSR-82)

   (JSR-75 File Connection)
   http://www.jcp.org/en/jsr/detail?id=75

[14] Sun Microsystems: Mobile Information Device Profile 2.0 (JSR-118)
   http://www.jcp.org/en/jsr/detail?id=118

[15] Sun Microsystems: Connected Limited Device Configuration 1.1 -
   Specification (JSR-139)
   http://www.jcp.org/en/jsr/detail?id=139

   http://nds2.forum.nokia.com/nds/ForumDownloadServlet?id=
   2343&name=Nokia%5FUIAPI%5FGuide%2Epdf


[18] Benhui
   http://www.benhui.net

[19] Sun J2ME Wireless Toolkit

   Paper, Version 1.1
   http://forum.nokia.com/main/0,6566,21,00.html?fsrParam=
   1-3-/main/0,,21,00.html&fileID=5287

[21] Forum Nokia: Device Details Nokia 6630
   http://forum.nokia.com/6630

[22] Sun Java Studio Mobility

[23] The Netbeans IDE
   http://www.netbeans.org

[24] Sun Microsystems: Signing and Verifying JAR Files
   http://java.sun.com/docs/books/tutorial/jar/sign/intro.html

[25] Legion of the Bouncy Castle: The Bouncy Castle Crypto package
   http://www.bouncycastle.org
[26] Forum Nokia: Known Issues In The Nokia 6600 MIDP 2.0 Implementation, Version 1.6
    http://www.forum.nokia.com/main/0,6566,148,00.html

[27] The ZigBee Alliance
    http://www.zigbee.org

Please note that the URLs noted below the reference entries have been valid at the time writing this document. They may be outdated in the meantime and therefore point to nowhere.