NTNU
Innovation and Creativity

The Improved Peer2Me Framework

A flexible framework for mobile collaboration

Tommy Bjørnsgård
Kim Saxlund

Master of Science in Computer Science
Submission date: May 2006
Supervisor: Alf Inge Wang, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

Peer2Me is a framework that enables developers to create collaborative applications for mobile phones using a network technology such as Bluetooth. This framework has been evaluated in the depth study performed fall of 2005: Evaluation of Peer2Me. The report shows that the framework contains several bugs and has limitations in certain areas. Before the framework is made available for other developers, these issues need to be resolved. The current version also requires a great deal of time to comprehend. It may therefore be necessary to redesign the architecture in Peer2Me in order to make it easier for other developers to utilize the framework. The tasks must be solved by using the Java 2 Micro Edition platform.

Assignment given: 2006-01-20
Supervisor: Alf Inge Wang, IDI

Abstract

This master thesis presents a new and redesigned version of Peer2Me, a framework for developing mobile collaborative applications on mobile phones. The first version of Peer2Me was designed and created by Carl-Henrik Wolf Lund and Michael Sars Norum in 2005, which was presented in their master thesis, *The Peer2Me Framework*, [31]. We evaluated their framework in our depthstudy [5], fall of 2005. The evaluation showed that the framework lacked some desired and necessary functionality, had some bugs and was a bit hard to use.

This thesis also describes the history of Peer2Me along with cental concepts regarding peer-to-peer networking in an mobile ad hoc environment. There are a lot of on going and finished projects that can be related to Peer2Me. We have chosen to investigate the most interesting and relevant projects, which are presented in Chapter 11, *State of the Art*. Since a redesign of the Peer2Me framework was necessary, we have performed a research in the most recognized architectural tactics, design patterns and architectural patters.

Before embarking on the task of designing the framework, a research in the latest technology was necessary. In our depthstudy [5], we had already performed such a research, so we only had to obtain the latest development in the related areas. Special attention was given to the Bluetooth wireless network technology.

All created packages, classes and interfaces are thoroughly described along with their roles in the framework. We felt that a mere description of the modules was not enough, so we wrote Chapter 16, *Design Decisions*, which discusses the different crossroads we faced with during development, and the path we chose.

To give the reader an impression of how the framework can be used, we also developed some applications that utilizes the new framework. Lastly we evaluated our work, compared the old and new framework, discussed the problems we encountered, answered our research questions and summarized the thesis.

All source code, javadoc and a functional, new version of Peer2Me are attached along with this report.

Preface

This master thesis presents the work and related results that Tommy Bjørnsgård and Kim Saxlund have contributed to the redesign of the Peer2Me framework in the spring of 2006. The Peer2Me project is related to the MOWAHS (Mobile Work Across Heterogenous Systems) project carried out at the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU).

Acknowledgements

We would like to thank Alf Inge Wang for his excellent advices and guidance during the spring of 2006.

Trondheim May 31st, 2006

Kim Petter Saxlund   Tommy Bjørnsgård

Contents

List of Tables x

List of Figures xi

I Introduction 1

1 Motivation 3
- 1.1 Problem definition 4

2 Project Context 5

3 Readers Guide 7
- 3.1 Chapter Description 7

II Research Methods 11

4 Research Questions 13
- 4.1 Research Questions 13

5 Research Methods 15
- 5.1 Research 16
- 5.2 Design 16
- 5.3 Implementation 17
- 5.4 Testing 17
- 5.5 Evaluation 18

6 Development Tools and Software 19
- 6.1 Development Tools 19
  - 6.1.1 Eclipse With Plugins 19
  - 6.1.2 MiKTeX 19
  - 6.1.3 Concurrent Version System 19
- 6.2 Emulators 20
  - 6.2.1 Sun Wireless Toolkit 20
  - 6.2.2 Sony Ericsson SDK 20

III Prestudy 21

7 The History of Peer2Me 23
- 7.1 MOWAHS 23
- 7.2 Work by Kirkhus and Sveen 23
  - 7.2.1 Spectre 24
- 7.3 Work by Lund and Norum 24
  - 7.3.1 Framework Prototype 24

7.3.2 Peer2Me, first version . . . 24
7.4 Our depth study . . . 25

8 Central Concepts **27**
8.1 Peer-to-Peer . . . 27
8.2 Mobile P2P Networks . . . 29
8.3 Mobile Ad Hoc Networks (MANET) . . . 29
8.4 Piconet . . . 31
8.5 Scatternet . . . 32

9 Communication and Collaboration **33**
9.1 Groupware and Computer Supported Cooperative Work . . . 33

10 State of the Art **37**
10.1 Bluecove . . . 37
10.2 Umbrella.Net . . . 37
10.3 BEDD . . . 39
10.4 Other projects . . . 40
10.4.1 JXTA . . . 40
10.4.2 JSR82 259 . . . 40
10.4.3 OBEX . . . 41
10.4.4 SyncML . . . 41
10.5 Summary . . . 41

11 Technology **43**
11.1 Mobile Phones . . . 43
11.2 Java 2 Micro Edition . . . 43
11.2.1 J2ME architecture . . . 44
11.2.2 Optional packages . . . 44
11.3 Wireless network technologies . . . 45
11.3.1 Bluetooth . . . 45
11.3.2 Wireless Local Area Network . . . 48
11.3.3 ZigBee . . . 48
11.3.4 Radio Frequency IDentification . . . 49
11.3.5 Wireless USB . . . 49
11.3.6 Wireless Firewire . . . 50

12 Software Architecture **51**
12.1 Stakeholders . . . 51
12.2 Quality of a System . . . 52
12.2.1 Quality Scenarios . . . 52
12.2.2 Quality Attributes . . . 53
12.3 Architectural Tactics . . . 53
12.3.1 Modifiability Tactics . . . 54
12.3.2 Usability Tactics . . . 55
12.3.3 Performance Tactics . . . 55
12.3.4 Availability Tactics . . . 56
12.3.5 Security Tactics . . . 57
12.3.6 Testability Tactics . . . 58
12.4 Design Patterns . . . 58
12.5 Behavioural patterns . . . 58
12.5.1 Observer pattern . . . 59
12.6 Creational patterns . . . 59
12.6.1 Singleton Pattern . . . 59
12.7 Structural Patterns . . . 59
12.7.1 Facade Pattern . . . 60

12.8 Architectural Pattern . . . 60
12.8.1 Layered Architecture Pattern . . . 60

IV Redesign of Architecture 63

13 Introduction 65

14 Requirements 67
14.1 Functional Requirements . . . 67
14.2 Quality requirements . . . 68
14.2.1 Modifiability . . . 68
14.2.2 Usability . . . 69
14.2.3 Testability . . . 69
14.2.4 Availability . . . 70

15 Architecture Design Decisions 73
15.1 A Pure P2P Model . . . 73
15.2 Layered Software Structure . . . 74
15.3 Communication protocol . . . 75
15.4 Messages . . . 75
15.5 Detecting Node loss . . . 77

16 Design Overview 79
16.1 Domain concepts . . . 79
16.2 High level general process scenarios . . . 80
16.3 Peer2Me and Java . . . 83
16.4 Peer2Me Package Overview . . . 84
16.5 The Framework package . . . 84
16.5.1 Class: Framework . . . 85
16.5.2 Interface: FrameworkSubscriber . . . 86
16.5.3 Class Diagram . . . 87
16.6 The Session package . . . 88
16.6.1 Class: Session . . . 88
16.6.2 Class Diagram . . . 89
16.7 The Service package . . . 90
16.7.1 Class: Service . . . 90
16.7.2 Class: HexBuilder . . . 90
16.7.3 Class Diagram . . . 90
16.8 The Group Package . . . 91
16.8.1 Class: Group . . . 91
16.8.2 Class Diagram . . . 91
16.9 The Node package . . . 92
16.9.1 Class: Node . . . 92
16.9.2 Class Diagram . . . 93
16.10The Network package . . . 93
16.10.1 Abstract Class: Network . . . 94
16.10.2 Abstract Class: NetworkNode . . . 94
16.10.3 Class: NetworkTimer . . . 94
16.10.4 Class Diagram . . . 96
16.11The Bluetooth package . . . 97
16.11.1 Class: Bluetooth . . . 97
16.11.2 Class: BluetoothListener . . . 97
16.11.3 Class: BluetoothConnectionHandler . . . 98
16.11.4 Class: BluetoothNode . . . 99
16.11.5 Class: BluetoothObjectPush . . . 99

16.11.6 Class: BluetoothPingListener 99
16.11.7 Class BluetoothPingConnectionHandler 99
16.11.8 Class BluetoothSearcher 99
16.11.9 Class: InstanceOfRemoteDevice 101
16.11.10 Class: MessageQueueProcessor 101
16.11.11 Class: Peer2MeHeaderSet 101
16.11.12 Class: PingQueueProcessor 102
16.11.13 Class Diagrams 103
16.12 The Message Package 105
16.12.1 Class: Message 106
16.12.2 Class: MessagePart 108
16.12.3 Abstract Class: FileObject 109
16.12.4 Class: SendFileObject 109
16.12.5 Class: ReceivedFileObject 109
16.12.6 Class: FileInfo 109
16.12.7 Interface: Serializable 110
16.12.8 Class: SendableNodeInfo 110
16.12.9 Class ObjectOutputStream 110
16.12.10 Class ObjectInputStream 110
16.12.11 Class LinkedMessageList 110
16.12.12 Class LinkedPingRecipientsList 110
16.12.13 Class Diagrams 111
16.13 The Exception package 115
16.13.1 FileNotFoundException 115
16.13.2 FrameworkNotInitializedException 115
16.13.3 GroupNotFoundException 116
16.13.4 NodeNotFoundException 116
16.13.5 LocalDeviceNotFoundException 116
16.13.6 HexConversionException 116
16.13.7 InvalidKeyException 116
16.13.8 UnknownKeyException 116
16.13.9 Class Diagram 117
16.14 The Log package 117
16.14.1 Class: Log 118
16.14.2 Class: LogElement 119
16.14.3 Class: LogElementDate 119
16.14.4 Class: LogElementType 119
16.14.5 Class Diagram 121
16.15 The Util package 121
16.15.1 Class: TextUtil 122
16.15.2 Class: FileHandler 122
16.15.3 Class Diagram 122

V Applications 123

17 Introduction 125

18 The Applications 127
18.1 Chat2Me 127
18.1.1 Walkthrough 127
18.2 File2Push 128
18.2.1 Walkthrough 129
18.3 ComplexMessageDemo 130
18.3.1 Walkthrough 130

VI Evaluation 133

19 Testing 135
19.1 Functional Requirements Results . . . 135
19.2 Quality Requirements Results . . . 135

20 Comparison of Old and New Peer2Me 139
20.1 Statistical Comparison . . . 139
20.2 Functional Requirements Comparison . . . 139
20.3 Differences that the Application Developer Experience . . . 141

21 Problems 145
21.1 Emulators vs Mobile Phones . . . 145
21.1.1 Performance . . . 145
21.1.2 Graphical User Interface . . . 146
21.1.3 Exception Handling and Debugging . . . 146
21.2 Understanding Bluetooth Limitations . . . 146
21.2.1 Master vs. Slave . . . 146
21.2.2 Number of Nodes . . . 147
21.2.3 Interference . . . 147
21.2.4 Security Issues . . . 147

22 Answers to Research Questions 149

23 Summary 151
23.1 Conclusion . . . 151
23.2 Further Work . . . 152
23.3 Short-term Goals . . . 152
23.3.1 Optimizing Network . . . 152
23.3.2 Configuration . . . 152
23.3.3 Scatternet . . . 152
23.3.4 Mobile Agents . . . 153
23.4 Long-term Goals . . . 153
23.4.1 Adopt new technology . . . 153
23.4.2 Up-to-Date . . . 154
23.4.3 Empirical Work and Applications . . . 154

VII Appendix 155

A Chat2Me Source Code 157

B File2Push Source Code 161

C ComplexMessageDemo Source Code 167

D Contents of Zip File 171

Glossary 173

Bibliography 175

List of Tables

8.1 Comparison of pure and hybrid model . . . 28

12.1 General availability scenario . . . 52
12.2 Sample availability scenario . . . 53

14.1 Functional requirements . . . 68

16.1 Available user defined OBEX headers . . . 102

19.1 Test results of the functional requirements . . . 136

20.1 Statistical data regarding Peer2Me . . . 139
20.2 Requirements that only exists in the old framework . . . 140
20.3 Requirements that only exists in the new framework . . . 140
20.4 Requirements that exists in both framework . . . 140
20.5 Comparing a chat application using the new Peer2Me framework to PanIm using the old framework and BlueChat from Ben Hui. . . . 142
20.6 Similar methods in new and old Peer2Me . . . 143

List of Figures

1.1 The principle of multihop ad hoc communication 3

5.1 The life-cycle of the waterfall method 16
5.2 The life-cycle of the waterfall method 17

8.1 Taxonomy of computer systems 28
8.2 Ad hoc networks taxonomy 30
8.3 A singlehop ad hoc network 30
8.4 Multihop ad hoc network 31
8.5 Scatternet comprising three piconets. 32

9.1 Clasifying groupware with the time-place matrix 34
9.2 Digital spheres intersecting 34

10.1 Typical umbrella.net schema 38
10.2 The GUI of the Umbrella.Net software 39

11.1 Overview of the Java enviroments 44
11.2 The Wireless USB topology 50

12.1 Example of a 3-layer layout in a fictitious application 61

15.1 The layered layout of Peer2Me 74

16.1 Domain concepts 80
16.2 Initialization of the Peer2Me framework 80
16.3 Search for other nodes 81
16.4 None of the nodes has been assigned a role 82
16.5 Node A sends a message to node B 82
16.6 Node B sends a message to node C 82
16.7 How the framework relates to J2ME 83
16.8 The packages of Peer2Me 84
16.9 Class Diagram of the Framework Package 87
16.10Class Diagram of the Session Package 89
16.11Class Diagram of the Service Package 90
16.12Class Diagram of the Group Package 92
16.13Class Diagram of the Node Package 93
16.14A simple hierarchical layout of network 93
16.15Class Diagram of the Network Package 96
16.16Node A searches and finds B and C 100
16.17Node A distributes findings 100
16.18All nodes can now establish connections with each other 101
16.19How messages are sent from one node to another. 102
16.20Class Diagram of the Bluetooth Package, part 1 103
16.21Class Diagram of the Bluetooth Package, part 2 104

16.22 Class Diagram of the Bluetooth Package, part 3 105
16.23 The classes in the Message package 106
16.24 Possible contents of a Message 107
16.25 The message queue. Before and after node A has been removed from the queue. 111
16.26 Class Diagram of the Message Package, part 1 112
16.27 Class Diagram of the Message Package, part 2 113
16.28 Class Diagram of the Message Package, part 3 114
16.29 An example of the inheritance of exception 115
16.30 Class Diagram of the Exception Package 117
16.31 The structure of the log system 118
16.32 Class Diagram of the Log Package 121
16.33 Class Diagram of the Util Package 122

Listings

11.1 Message header from the old Peer2Me framework, from `messageHeader.txt` . . . 46
12.1 Sample code showing singleton in use . . . 59
15.1 Sending two messages identified by an int value . . . 76
15.2 Receiving a message . . . 76
16.1 The Framework is created in an application . . . 85
16.2 The methods in the FrameworkSubscriber needed by Framework . . . 86
16.3 Example of creating a group and moving a node into it . . . 91
16.4 Codesnippet from Session.java where node is created . . . 92
16.5 The Bluetooth-class inherits the abstract Network-class . . . 94
16.6 The BluetoothNode-class inherits the abstract NetworkNode-class . . . 94
16.7 How NetworkTimer is created, and the ping-functionality is started . . . 94
16.8 Handling a received message in the MIDlet . . . 98
16.9 Example of creating and sending a simple message in the old framework . . . 107
16.10Example of creating and sending a simple message in the new framework . . . 107
16.11Example of creating and sending a complex message in the new framework . . . 107
16.12Retrieving info about a file . . . 109
16.13Example of adding an entry to the log . . . 118
16.14Example of retrieving entries from the log . . . 119
18.1 Necessary import declarations . . . 127
18.2 FrameworkSubscriber and Framework instance . . . 127
18.3 Initialize framework . . . 128
18.4 Search . . . 128
18.5 Send a message . . . 128
18.6 Receive a message . . . 128
18.7 Create a message and attache a file . . . 129
18.8 Set received files folder . . . 129
18.9 Receiving message containg file . . . 129
18.10Receiving chunks of message . . . 129
18.11Creating a complex message . . . 130
18.12Receiving a complex message . . . 130
20.1 How to implement the old framework . . . 141
20.2 How to implement the new framework . . . 141
20.3 How to send a message in the old framework . . . 142
20.4 How to send a message in the new framework . . . 142
A.1 Chat2Me source code . . . 157
B.1 File2Push.java . . . 161
B.2 LocalFileBrowser.java . . . 164
C.1 ComplexMessageDemo source code . . . 167

Part I

Introduction

CHAPTER 1

Motivation

Mobile phones and PDAs are getting more and more advanced. Mobile phones have gone from being simple phones to personal assistants, much like the first PDAs that entered the market. The usage areas for these phones are therefore increasing, and so is the public interest. Mobile phones can be used for much more than just making a phone call. Most recent mobile phones have calendars with scheduling possibilities, web browsers, e-mail support and lots of other applications. Many of these applications have been programmed using the Java 2 Microedition platform (J2ME), see [22] . Mobile phones are often equipped with several network technologies such as GSM, UMTS(Universal Mobile Telecommunications System), Bluetooth, IrDA and WLAN. These technologies make it possible to connect mobile phones to other devices in several ways. All of which have different advantages and disadvantages. Several mobile phones that interconnect using a network technology such as Bluetooth make up a Mobile Ad Hoc Network (MANET). The vast functionality of mobile phones combined with state of the art network technology makes them ideal as mobile collaborative devices.

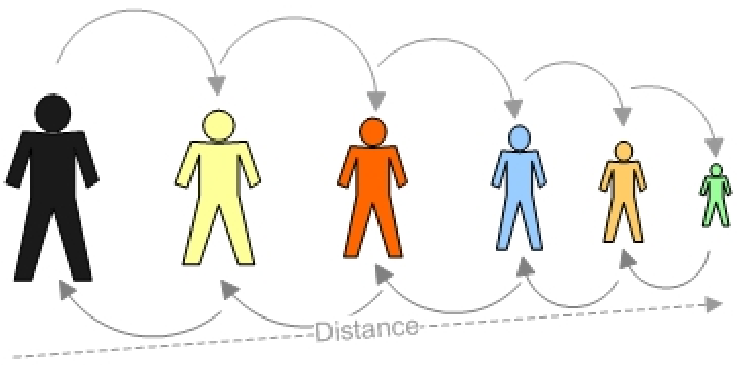


Figure 1.1: *The principle of multihop ad hoc communication*

Peer-to-Peer is a network that requires no infrastructure, which is perfect for mobile ad hoc communication. Peer-to-Peer requires no central server or router to handle the communication. Communication between mobile phones is most often spontaneous when Bluetooth or other short ranged network

technology is used. Ad hoc network connections can be utilized to the fullest with peer-to-peer technology. Examples of useful peer-to-peer applications are: synchronization of calendars, file sharing, contact card exchange, chatting, etc. There are also possibilities for more entertaining applications such as the PeerQuiz application we developed in our depth study [5].
Since no infrastructure is needed, these kind of applications are completely free to use as well. If a scatternet[1] can be implemented using Bluetooth technology, a person can theoretically chat with a person that is 2 kilometers away, which is far out of the range of a Bluetooth device's coverage area. This is illustrated in Figure 1.1.

Developers are often required to spend much time creating network modules for such mobile ad hoc network applications. Peer2Me is a framework that takes care of this, so that the developer can focus on other important part of the application, such as the logics and graphical user interface. In our depthstudy [5], we evaluated the framework by developing two advanced applications. The result showed that the framework was not mature or good enough to be adopted on a broad scale. The framework was hard to adopt, contained some bugs and lacked certain wanted functionality. Therefore, a redesign of the framework was necessary.

1.1 Problem definition

The main goal of the Peer2Me project is to distribute the framework as a digitally signed freeware over the Internet. In order for this to be successful, the framework has to be thoroughly tested and without bugs. A substantial amount of sensible functionality is needed for the framework to become popular.

The objective of this master thesis is to improve the Peer2Me framework, based on the result of our depthstudy performed fall of 2005 [5]. We will repair several bugs concerning stability and availability, and add new functionality, which will greatly improve the flexibility and usability of the framework. In order to do this, we will redesign the framework. Even though we will implement a new design, old parts of the framework can be reused where we find it appropriate.

We will also look into new versions of Java and/or Bluetooth which can be utilized to improve the framework.

[1]A scatternet exists if a node is connected to two or more separate networks at the same time. For more info, read Section 8.5

CHAPTER 2

Project Context

This project is a part of MOWAHS, *MObile Work Across Heterogeneous Systems*, which is a basic research project performed in cooperation with the Software Engineering and the Database Techonology groups of The Department of Computer and Information Science (IDI) at The Norwegian University of Science and Technology (NTNU).

MOWAHS' superior goals are:

1. Helping to understand and to continuously assess and improve work processes in virtual organizations
2. Providing a flexible, common work environment to execute and share real work processes and their artifacts on a variety of electronic devices (from big servers til small PDAs).
3. Disseminating the results to colleagues, students, companies, and the community at large.

As Lund and Norum stated in [31], the Peer2Me framework is mainly contributing to the second goal of MOWAHS. Our work will continue supporting this goal. The Peer2Me framework will be available on the web for anyone who has interest in the project.

This thesis will also contribute to the third goal, but it will focus more on software engineering than collaboration technology.

CHAPTER 3

Readers Guide

The contents of this report vary a lot, both with respect to focus and approach. Some parts of the report might be more interesting to some readers, than others. In order to increase the readability of the report, we will therefore present a brief outline of each chapter.

If you are not interested in reading the whole report, you should identify yourself with one of the three following categories:

Readers interested in the problem domain: Should read Chapter 1 along with Part III. Some readers may read Chapter 15 as well. Part III describes essential concepts and technologies along with a chapter, state of the art, presenting other projects related to the problem domain.

Developers interested in writing applications: Should take a look at the example applications in Part V. To get background information on how the framework is designed and its possibilities, he/she could also read Part IV.

Developers interested in improving the framework: Should read and understand Part IV, Part V and Part VI. These parts present all essential information about the framework beginning with how it is designed and implemented, how to develope simple applications using the framework, and an evaluation of the implementation.

3.1 Chapter Description

This section outlines the chapters in the report.

- **Part I - Introduction**

  Chapter 1 - Motivation This chapter describes the motivation behind the project along with the problem definition.

  Chapter 2 - Project Context This chapter describes the context of the project.

  Chapter 3 - Readers Guide This chapter presents an outline of the report describing the chapters.

- **Part II - Research Methods**

Chapter 4 - Research Questions This chapter described the research questions that is derived from the problem definition and motivation.

Chapter 5 - Research Methods This chapter presents the research methods used in this project.

Chapter 6 - Development Tools and Software This chapter presents the tools and software used to create this report and the framework.

- **Part III - Prestudy**

Chapter 7 - The History of Peer2Me This chapter presents a chronological description of how the framework initially was drafted.

Chapter 8 - Central Concept This chapter deals with the central concepts related to our problem domain. These concepts should be understood in order to grasp the background and intention behind the framework.

Chapter 9 - Communication and Collaboration This chapter discusses the basics behind Groupware and Computer Supported Cooperative Work (CSCW).

Chapter 10 - State of the Art This chapter presents similar projects and solutions as the Peer2Me, and compare these.

Chapter 11 - Technology This chapter presents an update on relevant technology, discussing advantages and disadvantages.

Chapter 12 - Software Architecture This chapter describes what software architecture is, discusses architectural tactics and presents relevant design patterns.

- **Part IV - Redesign of Architecture**

Chapter 13 - Introduction This chapter introduces Part IV.

Chapter 14 - Requirements This chapter describes the requirements for the new Peer2Me framework.

Chapter 15 - Architecture Design Decisions In this chapter we presents the main issues we met during development, and how we dealt with them.

Chapter 16 - Design Overview This chapter presents the actual design of the new framework, going in detail on the packages and the classes. It also presents the domain concepts.

- **Part V - Applications**

Chapter 17 - Introduction This chapter introduces Part V.

Chapter 18 - The Applications Here we present three applications that utilize the new Peer2Me framework.

- **Part VI - Evaluation**

Chapter 19 - Testing Compares the actual implementation to the requirements in Chapter 14.

Chapter 20 - Comparison of Old and New Peer2Me This chapter compares the old version of the framework to the one we have implemented, discussing improvements and new functionality along with simple statistics.

Chapter 21 - Problems This chapter discusses problems and difficulties we came across during this project.

Chapter 22 - Answers to Research Questions This chapter answers the research questions presented in Chapter 4.

Chapter 23 - Summary This chapter gives a conclusion to the project, discussing further work with short and long term goals.

- **Part VII - Appendix**

Glossary A simple glossary elaborating difficult terms.

Appendix A Chat2Me source code.

Appendix B File2Push source code.

Appendix C ComplexMessageDemo source code.

Appendix D Contents of zip file.

- **Bibliography**

Part II

Research Methods

CHAPTER 4

Research Questions

This chapter outlines the questions we seek to answer.

4.1 Research Questions

In every development project, the process of creating software includes major challenges and trade-offs. These trade-offs will aid the development project to completion, but will often involve short-cuts and result in instability and lack of performance. It is therefore important in every development project, that the product is thoroughly tested and completed before the first official release.

As Peer2Me is a framework to be used by other developers, it is important that it is easy to comprehend and utilize. As mentioned in Chapter 1, a redesign of the architecture of Peer2Me is appropriate. In addition to removing the bugs, new functionality should also be introduced in the new version of the framework.

The questions given below are the central aspects which we will seek to enlighten in this project.

1. Will a redesign make the framework easier to adopt, when developing mobile ad-hoc applications in a J2ME development environment?
   (a) Can a pure peer-to-peer network be implemented in the new framework?
   (b) Is it possible to implement transfer of binary data in the new framework and what consequences will this yield?
   (c) Will developers use less time to learn and create applications with the new framework compared to the old one?
   (d) Will a redesign of the framework reduce the number of codelines, memory usage and dependencies in applications that uses the new framework compared to the old one?
2. Will implementation of extra functionality make the framework more flexible and attractive?
   (a) What sort of functionality would add most value to the framework?

   (b) Will the extra functionality make the new framework incompatible with mobile phones that the old framework supports?

3. What sort of impact would new technology or updates in existing technology have on the Peer2Me framework?

   (a) Will the technology make mobile ad hoc collaboration more efficient in terms of discoverytime, range and transfer rates?

CHAPTER 5

Research Methods

This chapter presents how the research and development in this master thesis will be performed. There are many methods that can be applied to a software development process, and some of them are presented in this chapter. Basili describes three common research methods that are relevant for software development in *Experimental Software Engineering Issues: Critical Assessment and Future Directions* [2]:

The engineering method (Scientific): Observe existing solutions, propose better solutions, build/develop, measure and analyze, and repeat the process until no more improvements appear possible. This method is an evolutionary approach were the software being developed can go through experimental phases in order to find the best solution. This method can also involve analysis of older versions of the software so that improvements can be made. This method is typically used to find better methods for structuring large systems. Analysis and measurement is crucial for the success of this method.

The empirical method (Scientific): Propose a model, develop statististical/qualitative methods, apply to case studies, measure and analyze, validate the model and repeat the procedure. This approach is classified as a revolutionary method which can either propose a model based on an existing one, or suggest an entire new model. This method is typicallly used when comparing a new technology against an old technology. Analysis and measurement is crucial for the success of this method as well.

The mathematical method (Analytic): This method is based on mathematical and formal methods for doing experiments. The formal method is compared with empirical observation to get results. The mathematical method is usually used to find better formal methods and languages.

We have chosen to use *the engineering method* since it seems like the best suited approach for our project. The method is based on an evolutionary approach which is representative for Peer2Me. The method also involves analysis of older versions, which in our case, is the old version of Peer2Me. Our goal is to develop a new version. We expect to encounter unforseen obstacles during development, which might result in change in design. Since we have chosen to use this method, we will test the framework during development to see if it satisfies the requirements we have set. There are many ways of solving a problem and we hope to find out the best solution by testing different implementations. In order to do this, we will have to identify the different solutions to the problems as they appear.

5.1 Research

Before answering the research questions by redesigning the Peer2Me framework, background knowledge about the old Peer2Me framework is essential. This is done by reading the Peer2Me documentation found in the master thesis by Lund and Norum[31], and getting an overview of the available technologies. Performing a thorough research will give good insight in the components that makes up the framework, and how they relate to each other.

5.2 Design

There are primarily two different approaches we can use to design the framework: The waterfall method or the evolutionary prototyping method. The waterfall method has five stages as illustrated in Figure 5.1. The strength of this method is that none of the stages needs to be repeated and planning and design can be done early in the process. The method is well suited for technically weak or inexperienced developers. Since the model follows this rigid pattern it is also highly unflexible.

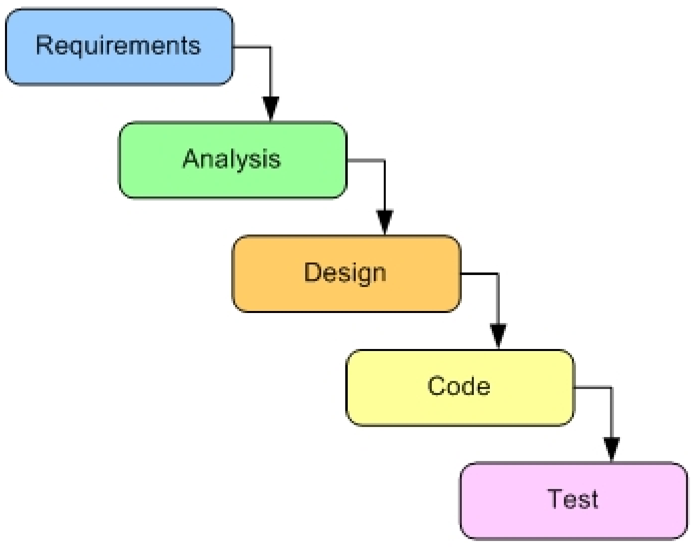


Figure 5.1: *The life-cycle of the waterfall method*

The evolutionary prototyping method consists of stages that can be repeated an unlimited number of times, see Figure 5.2. The model is less rigid and is perferct when the requirements are changing or the application area is hard to grasp. Since steps can repeated many times, it is hard to tell how long the development will take. It is also unknown how many iterations that is required to get a satisfactory result.

We have chosen to use the evolutionary prototyping model as a guide during development in this project. The problem domain is unknown in some areas and we are unsure if we are able to predict how the system will react on a specific implementation. It will therefore be necessary to go through a couple of iterations in order to have a satisfactory end result. Since we are going to redesign the framework, we will use the evaluation we performed in our depthstudy [5] as a guide to find the areas of improvement. This will result in a set of both functional and non-functional requirements. The requirements are described in Chapter 14.

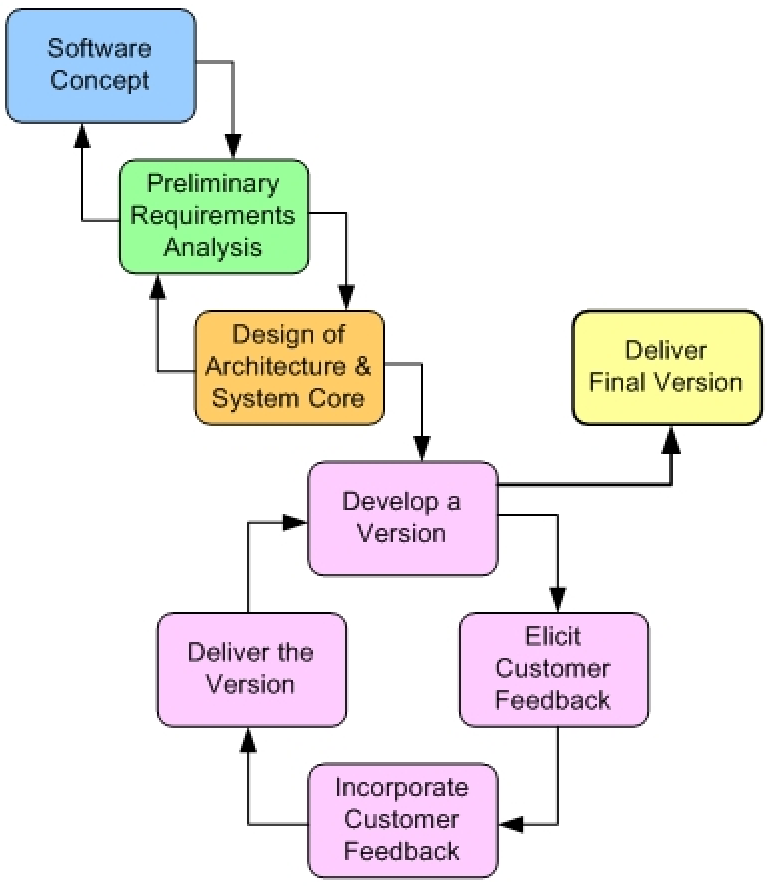

Software Concept
Preliminary Requirements Analysis
Design of Architecture & System Core
Deliver Final Version
Develop a Version
Deliver the Version
Elicit Customer Feedback
Incorporate Customer Feedback

Figure 5.2: *The life-cycle of the waterfall method*

5.3 Implementation

The redesign of the framework will be done with the latest available Java version as an incremental process laying out the base structure and adding functionality. The new framework will be based on the same concepts as the previous version, but with a whole new implementation. During the implementation all problems and issues will be noticed, and if applicable, used to answer the research questions described in Chapter 4.

5.4 Testing

Since we are using the evolutionary prototyping model, we will perform testing for each increment of the framework. The testing will reveal if our solution was satisfactory. The testing will be performed on the emulators on desktop computers and on mobile phones. During development in our depthstudy [5], we discovered that even though an application runs perfectly on the emulator, it might not run at all on mobile phones. It is a tedious process to test on the mobile phones, since we will have to first transfer the application to several mobile phones, using either Bluetooth or a cable and then install the application and run it. This will not be done for each small change, but for more larger changes.

5.5 Evaluation

We will evaluate the Peer2Me framework by testing it with the applications described in Part V. This way we can find out if the functional requirements have been fulfilled. We will also use the applications to test the quality requirements.

Comparing the new and the old framework will reveal differences and point out improvements, and how this affects the application-developer. An empirical comparison will also be made.

During the development we will probably encounter obstacles and problems. These will also be discussed in the evaluation, and how it has affected the framework.

All of the above will provide us with enough material to answer the research question. Lastly, a summary of the project will be presented along with suggestion for further work and short/long-term goals. The evaluation is described in Part VI.

CHAPTER 6

Development Tools and Software

This chapter presents the different tools used to create this report and the framework.

6.1 Development Tools

Peer2Me is a framework written in Java and the goal of this master thesis is to redesign and improve the framework. This section presents the tools used.

6.1.1 Eclipse With Plugins

Eclipse [13] is a an open source development platform with lots of available plugins for different purposes, e.g. writing in LaTeX, creating applications for mobile phones and finding code statistics.

TeXlipse TeXlipse is a plugin for writing LaTeX documents in Eclipse. It includes features like syntax highlighting, command completion and bibliography completion [16].

EclipseMe EclipseMe [15] is a plugin for developing J2ME MIDlets in Eclipse. A Java Wireless Toolkit must be installed on the system for the plugin to work.

Metrics Metrics is a plugin that generates code statistics for any given project [19]. The statistics it generates are very thorough and only small portions of it will be used in this thesis.

6.1.2 MiKTeX

MiKTeX is an implementation of TeX and related programs for Windows on x86 systems [39]. The MiKTeX distribution contains many features including the pdfTeX compiler which generates a pdf document. The compiler is used to produce this document.

6.1.3 Concurrent Version System

In order to keep track of versions of code and documentation, we have used a Concurrent Versioning System (CVS). NTNU has a Linux server with CVS support which we have used in this master thesis. The system also makes it easy to work from any computer that has Eclipse and the necessary plugins.

6.2 Emulators

Applications made in J2ME need to be tested. The applications can run on mobile phones, but it is a tedious process to do for each iteration. Therefore, emulators can be used instead. There are several emulators available and each mobile phone producer has its' own toolkit. In our depth-study [5] we only used the Sun Wireless Toolkit. During the master thesis we will also use the Sony Ericsson Wireless Toolkit. The reasoning for this decision is given in Section 6.2.2.

6.2.1 Sun Wireless Toolkit

The Sun Wireless Toolkit [36] is a standard toolkit that can be used to test applications that are based on J2ME's Connected Limited Device Configuration (CLDC) and Mobile Information Device Profile (MIDP). During the testing phase in the depthstudy [5], we found limitations with this toolkit. It is possible to run several instances of the toolkit so that instances can communicate with each other using Bluetooth. This is sufficient enough for the Peer2Me framework itself, but not for the applications we developed. Only one instance can access the emulated recordstore, filesystem and PIM information database. This proved to be a huge drawback when testing the PeerShare application, see Part V in our depthstudy [5].

6.2.2 Sony Ericsson SDK

The Sony Ericsson software development kit is a wireless toolkit that can be used to emulate Sony Ericsson mobile phones that support Java ME technology [1]. The kit can emulate the following phones: W800, W600, W550, Z520, K750, K600, K300, J300, Z800, V800, S700/S710, Z500, K700, Z1010, K500, K508, F500i, P900, P910, Z600/Z608, T630-T628, T637 and T610 Series (T610, T616 and T618). P900 and P910 is not supported. The SDK can run applications that uses both MIDP 1.0 and 2.0. In contrast to Sun's wireless toolkit, the Sony Ericsson toolkit can run several instances of an application using emulations of different phones so that each phone has a recordstore, filesystem and PIM database. This enables us to test the applications with the emulator, where we previously had to deploy the applications on actual phones. However, using this SDK will only test how the framework and applications will work on Sony Ericsson mobile phones. Therefore, the framework must be tested using both Sun's wireless toolkit and the Sony Ericsson SDK as must as possible to ensure that the software will run on different hardware.

Part III

Prestudy

CHAPTER 7

The History of Peer2Me

This chapter will provide background information on how the Peer2Me initially was created, its preliminary projects and intentions.

7.1 MOWAHS

As mentioned, this project is a part of MOWAHS, *MObileWork Across Heterogeneous Systems*, which is a basic ongoing research project in cooperation with the Software Engineering and the Database Techonology groups of The Department of Computer and Information Science (IDI) at The Norwegian University of Science and Technology (NTNU). MOWAHS' goals are threefold. The second goal states that *MOWAHS should provide a flexible, common work environment to execute and share real work processes and their artifacts on a variety of electronic devices.* It is this goal Peer2Me aims to support.

The MOWAHS project was started in 2001 and was initially planned for 4 years with 2.5 PhD students and 1 postdoc researcher on average per year. The project has been extended and is evaluated on a yearly basis.

7.2 Work by Kirkhus and Sveen

Two former students at The Norwegian University of Science and Technology (NTNU), *Lars Kirkhus* and *Anders R. Sveen* wrote in 2003 a depthstudy aimed to examine the use of mobile devices for spontaneous collaboration. This was the start of Peer2Me, as these two student started testing Java's new API for Bluetooth; the JSR82 API, described in Section 11.2.2. They started testing the software using only computers since no mobile devices were available at that time. They focused on evaluating discovery times and transfer rates. Doing this revealed possible problems due to the design of Bluetooth. For more information, please read their depthstudy, *An examination of mobile devices for spontaneous collaboration*[25].

7.2.1 Spectre

Kirkhus and Sveen also performed their master thesis within the MOWAHS-project, creating the first draft of a framework for supporting mobile ad-hoc communication. They called the framework *Spectre*, and it was constructed of four parts:

- **Application Interface** - This is where the application asks the framework to perform tasks.
- **System** - It contains the main logic of the framework.
- **Transport Interface** - The interface and implementation that enables the framework to do the low level communication with other devices.
- **Domain** - The classes that describes the environment and information the framework operates within, such as Node etc.

They developed several applications to run on their framework, concluding that developing a framework for mobile collaboration is a difficult task requiring good planning. The obstacles that lies within new technologies can also be hard to overcome. The master thesis was written in 2004, and for more information please read their master thesis, *MOWAHS - Mobile Collaboration Framework*[26].

7.3 Work by Lund and Norum

When Kirkhus and Sveen finished their master thesis, they left NTNU leaving the MOWAHS project behind. The project was now undertaken by *Carl-Henrik Wolf Lund* and *Michael Sars Norum* as a part of their depthstudy. They also explored the domain of collaborative applications using mobile devices, and presented a set of requirements for a new improved framework focusing on network independency. Along with the actual plans of a new framework, they described technologies and theoretical principles behind peer-to-peer computing.

7.3.1 Framework Prototype

As mentioned, Lund and Norum drafted the outline of a new and improved framework making it network independent. They implemented a prototype of the framework including a package for communicating using Bluetooth. Their framework was built on 4 main concepts:

- **Framework** - The core entity and interface between the application and rest of the system.
- **Node** - A logical representation of a peer, i.e. a mobile phone running the framework.
- **Network** - An abstraction of the network layer accessed only by the framework instance.
- **Service** - The description and identificator of an application running on the framework.

They concluded their framework was quite unique, but it had difficulties running some of the test applications on a mobile phone. Their prototype was only able to connect 2 mobile phones. For more information, please read their depth study, *A Framework for Mobile Collaborative Applications on Mobile Phones*[30].

7.3.2 Peer2Me, first version

In 2005 Lund and Norum continued their work within the MOWAHS-project as part of their master thesis. This thesis is based on their depth study mainly refining the framework, now renaming it Peer2Me. Peer2Me is an abbreviation for peer-to-peer (P2P) and Java 2 Micro Edition (J2ME). Using the *engineering approach*, they evaluated their own work and redesigning the framework basing it on a *peer-to-peer* *hybrid model*. The engineering approach is described in this document in Chapter 5, and the P2P hybrid model is covered in Section 8.1.

Their version of the Peer2Me framework defined a rigid master/slave setup, routing all information traffic through the master-peer. They defined domain concepts such as a *group*, a *node* and a *service*, and managed to connect many nodes, not just two as in the prototype. It was also only possible to send textmessages, and all the messages were sent through the master node.
To evaluate their framework, they created three applications focusing on different parts of the framework.

For more information on Lund and Norums framework, please read *The Peer2Me Framework. A Framework for Mobile Collaboration on Mobile Phones*[31].
It is this project, we are redesigning Peer2Me. A comparison of their and our version of the framework can be found in Section 20.

7.4 Our depth study

Continuing the work of Lund and Norum, we evaluated their version of the framework in our depth study, creating two applications. The applications, named *PeerQuiz* and *PeerShare*, focused on utilizing as much of the framework as possible. As mentioned in Section 7.3.2, Peer2Me is based on a hybrid peer-to-peer model which works fine with PeerQuiz, since this is a quiz-game with *one* defined leader, namely a master node. We tested the application with several nodes, and it did function as planned, but since it is not possible to send other data than text between the nodes, the quiz was limited. Sending pictures, sound etc. would have improved the application.

PeerShare is a filesharing application with no defined roles, it would therefore have been ideal if the framework was based on a pure peer-to-peer model. The ability to send binary data was also sorely needed in this application, since the only type of file the application could send and receive, were text files.

As a part of our evaluation, we answered 4 research questions:

1. Can a developer with some experience in Java, easily adopt the Peer2Me framework as a utility for developing applications for mobile phones?
   The framework is not easy to start using, but it saves the developer a lot codeline in the application using the framework.

2. Which bugs exists in the framework and what kind of impact do they have for development?
   We found five significant bugs. These are listed in Chapter 22 in our depthstudy [5].

3. How can we improve the framework?
   To improve the framework, we must fix the bugs from the answer to research question 2, along with adding functionality as described in the answer to research question 4.

4. How can we add functionality and value to the framework?
   The framework can never get enough functionality to suit every situation, but implementing all sorts of functionality would make it too large and impossible to use on all devices. A list with suitable functionality is found in Chapter 22 in our depthstudy [5].

These research questions evaluated most of the framework through our applications pointing at important issues. The answers to these questions along with an update on new technology and state-of-the-art software can be found in our depth study, *Evaluation of Peer2Me*[5].

.

CHAPTER 8

Central Concepts

This chapter explains the central concepts about peer-to-peer (P2P) computing systems, and how they can be used on mobile devices, without an infrastructure. This chapter also explains some basic concepts about *groupware and Computer Supported Cooperative Work* (CSCW).

8.1 Peer-to-Peer

Computer systems can either be defined as *centralized* or *distributed*, as found in *Distributed Systems: Concepts and Design* [10]. Mainframes and workstations are examples of centralized systems; the system can operate on its own without the aid of another computer or a network. Distributed systems can be thin clients or terminal computers; computers that require an other computer in order to function properly. Distributed systems are then classified into two architectures; the typical server-client model and the P2P model, see Figure 8.1.
Wikipedia defines P2P as:

> ”A peer-to-peer (or P2P) computer network is a network that relies on the computing power and bandwidth of the participants in the network rather than concentrating it in a relatively few servers. P2P networks are typically used for connecting nodes via largely ad-hoc connections” [44]

Lund and Norum [31] has recognized these advantages of P2P networking:

Decentralization: There is no significant load on one particular node, which could create a bottleneck in the network.

Capacity: Since a network connection between two peers does not depend on a central server, bandwidth, storage and processing power on the edge of the network is better utilized.

Independency: Each node is independent of a central server.

Extensibility: Peers can create a network, or easily join an existing network and increase its value.

Configuration: All peers are equal in terms of functionality and role, therefore, the network is self-configurable.

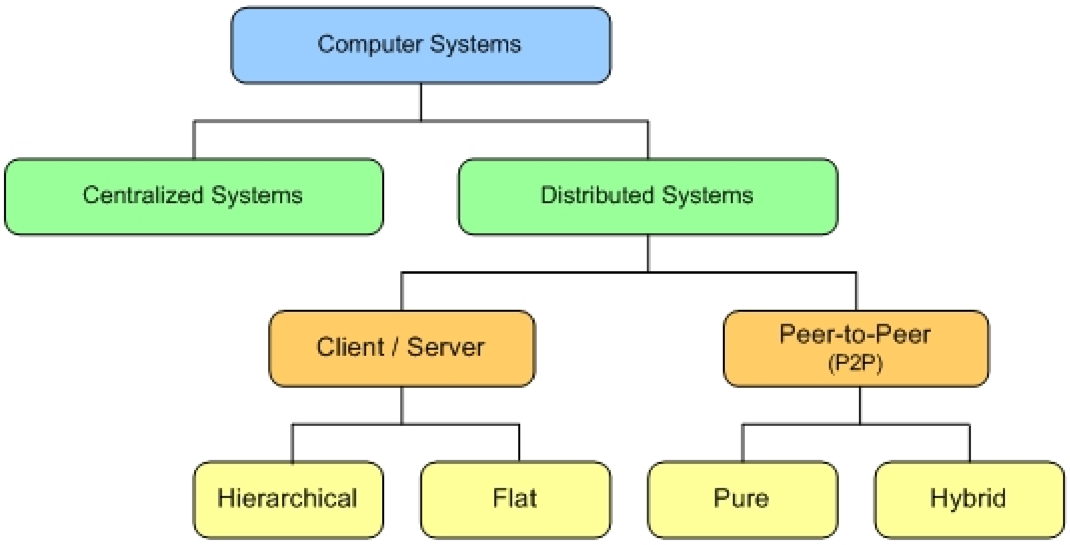


Figure 8.1: *Taxonomy of computer systems*

Fault tolerance: No single point of failure.

The resources can be shared between nodes, also called peers, and a peer can be either server or client. As defined by *Wikipedia* [44], P2P is divided into three categories:

Pure P2P: All peers can act as both clients and servers with no central server.

Hybrid P2P: Needs a central server which keeps information on peers and responds to nodes requesting information.

Mixed P2P: A combination of pure and hybrid P2P.

Table 8.1 compares the pure and hybrid P2P networks.

Pure P2P	Hybrid P2P
• All peers have the same responsibility • No central entity for management and coordination • Can easily loose one peer with no loss of functionality • Requires complex routing and location protocols	• One or more central entities responsible for providing services to other peers • Central entities are contacted by peers

Table 8.1: *Comparison of pure and hybrid model*

There are several applications freely available to the public that use P2P technology. Among the most popular are filesharing applications such as KaZaA and Direct Connect. Bittorrent are the latest member

to this family. Instant messaging applications such as MSN Messenger and ICQ are also quite popular P2P applications.
It is important to keep in mind that applications based on P2P networking have very high network externalities; it would be useless if there are very few users, but the value of the application increases rapidly as more people start using them.

8.2 Mobile P2P Networks

Mobile P2P networks should preferably be based on a *pure* P2P model, so that the peers can discover each other without the use of a central node. This can be implemented in two ways:

With infrastructure: Base-station-based networks like GSM or UMTS.

Without infrastructure: Networking using mobile ad hoc networks like WiFi, Bluetooth or Infrared.

Creating applications based on the pure P2P model, includes a significant amount of challenges. Forman and Zahorjan defines three categories of challenges for mobile computing which are highly relevant for mobile P2P systems in their article "*The challenges for mobile computing*" [12]:

Communication: Varying bandwidth, interference, security, users outside coverage area, frequent disconnections, delays and integration of heterogeneous network.

Mobility: Difficult to address devices and location-dependent.

Portability: Size, weight, limited battery power, exposed to weather, must operate in noisy surroundings.

Of the issues mentioned above, disconnections are probably the biggest challenge when designing P2P applications for mobile devices. The devices are constantly moving around and the connection may be lost at any time. This will often result in incomplete data transfers between peers, which must be accounted for in order to create solid and useful applications.

8.3 Mobile Ad Hoc Networks (MANET)

Mobile devices equipped with network technologies such as Bluetooth or Infrared can constitute a wireless personal area network (WPAN). According to *Wikipedia*, the definition of a personal area network (PAN) is:

> "A personal area network (PAN) is a computer network used for communication among computer devices (including telephones and personal digital assistants) close to one person. The devices may or may not belong to the person in question. The reach of a PAN is typically a few meters. PANs can be used for communication among the personal devices themselves (interpersonal communication), or for connecting to a higher level network and the Internet (an uplink). Personal area networks may be wired with computer buses such as USB and FireWire. A wireless personal area network (WPAN) can also be made possible with network technologies such as IrDA and Bluetooth." [45].

Several WPANs can make up a mobile ad hoc network (MANET). Wikipedia defines MANET as:

> "A mobile ad-hoc network (MANET) is a self-configuring network of mobile routers (and associated hosts) connected by wireless links the union of which form an arbitrary topology. The routers are free to move randomly and organize themselves arbitrarily; thus, the network's wireless topology may change rapidly and unpredictably. Such a network may operate in a standalone fashion, or may be connected to the larger Internet" [42].

Mobile phones use cellular technology relying on systems such as GSM and UMTS. These systems relies on a underlying infrastructure. Ad hoc mobile networks do not need such infrastructure, which has its advantages. "*When Peer-to-Peer comes Face-to-Face: Collaborative Peer-to-Peer Computing in Mobile Ad hoc Networks*" by Gerd Kortuem et.al. [27] lists three of these advantages:

No infrastructure required: Since no infrastructure is required, the network can be deployed spontaneously when needed anywhere.

Self-organization: In a wired network, the topology is determined by the cables that connect the nodes to each other. In an MANET, the network is created as soon as two nodes are within each others PAN. This means that the network is continuously reconfigured as mobile devices enters and exits each other's PANs.

Fault tolerance: Since there are no infrastructure, the mobile devices cannot fail because of a base station, which is the case in networks that rely on GSM communication. The only way a MANET can fail is if one node fails, but since the network is based on P2P communication, the network can easily be self-reconfigured.

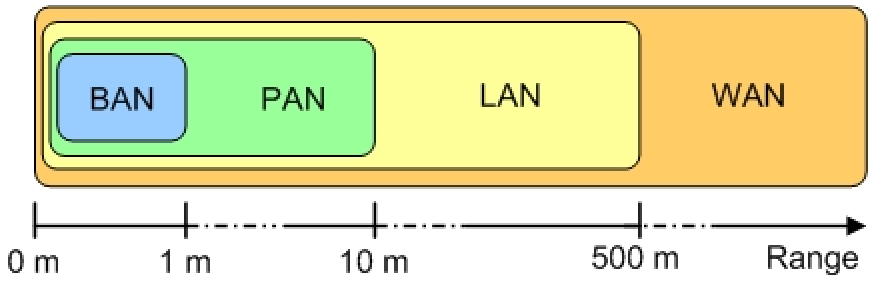


Figure 8.2: *Ad hoc networks taxonomy*

Figure 8.2 shows the taxonomy of ad hoc networks, where the horizontal axis describes the range of each technology. They are Body(BAN), Personal(PAN), Local(LAN) and Wide Area Network(WAN). Peer2Me is based on Bluetooth-technology, and is therefore used in PANs.

The information shared between nodes in an ad hoc network may follow different paths depending on how many parties involved. It can roughly be divided into two groups:

Singlehop: Only two nodes are involved in a singlehop network, and the communication is performed directly between the two peers, as Figure 8.3 shows.

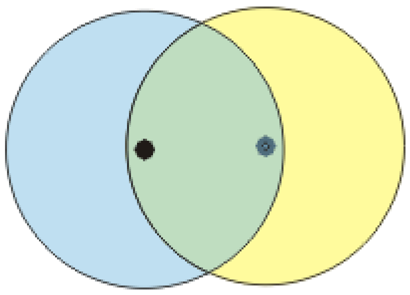


Figure 8.3: *A singlehop ad hoc network*

Multihop: A multihop configuration require several nodes, where some nodes are out of direct contact with each other. The communication between these nodes are then forwarded by other intermediate nodes in the network. Figure 8.4 illustrates the multihop where the black lines are the communication paths.

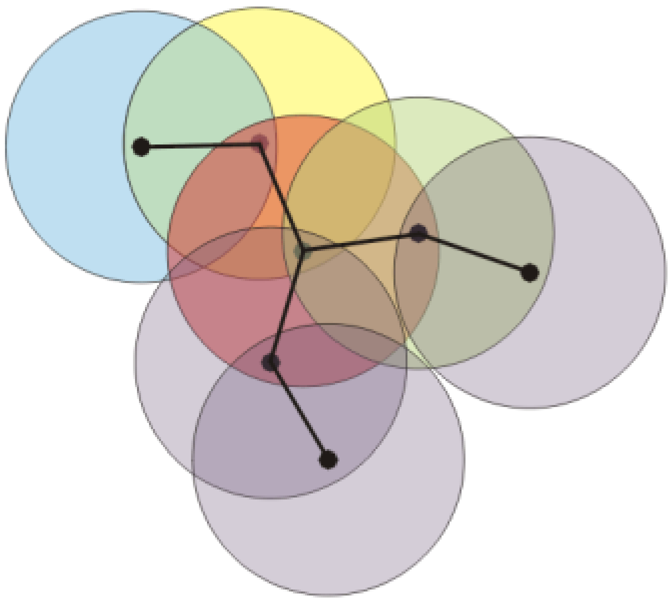


Figure 8.4: *Multihop ad hoc network*

8.4 Piconet

Peer2Me uses a form of PAN to communicate, and a Bluetooth PAN can consist of up to eight devices where there must exist one master, while the other nodes acts as slaves. This is called a *piconet*. A piconet can start of with singlehop including only two nodes, and then be extended with new nodes converting to multihop. It is important to keep in mind that the limit of eight devices means eight **active** devices, while a masternode theoretically can keep direct track of up to 255 **parked** devices. The master can "swap out" active slaves for parked slaves to manage piconets for situations that require a large number of connected devices, such as providing data services to people in high-population areas.

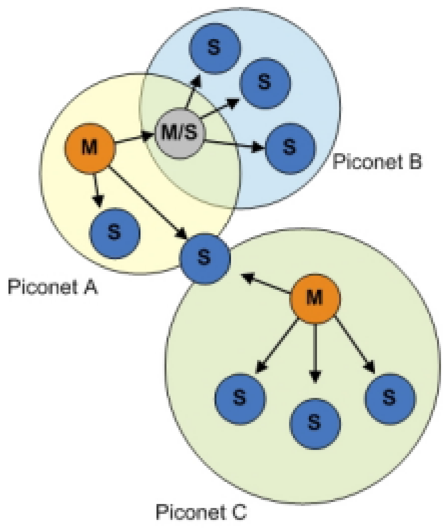


Figure 8.5: *The three piconets A, B and C together form a scatternet. m=master, s=slave, m/s=master and slave.*

8.5 Scatternet

A scatternet exists if a node is connected to two or more separate piconets at the same time, as illustrated in Figure 8.5.

However, this means that a message from one node may have to be forwarded by several nodes, which raises security concerns. The problem with the current Bluetooth implementation is that it does not allow a node to function as both a master and a slave simultaneously, but sequential.

CHAPTER 9

Communication and Collaboration

Communication and collaboration among humans is both a natural and difficult part of our civilization. Most of us change our behaviour in different contexts and we rarely give it much thought, it happens automatically. In todays society, technology is becoming increasingly pervasive, affecting the way we communicate with each other. Mobile phones , instant messaging and e-mails are more and more replacing common face-to-face communication and are impersonalizing the art of conversation.
Much research has been carried out in analysing how humans collaborate without technical devices in order to create the same context with the use of computers, hand held devices and other communication tools. This has proven to be a difficult task. Transferring real life to a digital world is hard due to many human factors which computer cannot reproduce. This can be informal chance-encounters by the coffee machine, soothing factors in a persons voice etc.

9.1 Groupware and Computer Supported Cooperative Work

In order to aid the transition between real life and the digital world as described above, much research in the field of Computer Supported Cooperative Work, CSCW, has been undertaken. The term CSCW was introduced by *Irene Greif* and *Paul Cashman* in 1984 at a conference attended by researchers and developers examining how people work together in groups and how technology can support them. This is described in a report by L.J.Bannon [29].
There are two main groups of cooperations, *informal spontaneous* and *formal planned* cooperation. Spontaneous, informal cooperation will often give greater results than planned meetings. When people start working together outside a formal setting, no special protocol is enforced, which sets out a loose and creative atmosphere. Groupware are computerized systems that have been created to support CSCW. These systems can be classified in a time-space matrix illustrated in Figure 9.1.

In one dimension, the matrix distinguishes same time (synchronous) cooperative work from different time (asynchronous). The other dimension distinguishes same place from different place cooperative work. Using this matrix, groupware systems can be classified by placing it in the quadrant or quadrants they support. When developing groupware, the biggest challenge lies in systems classified as either *different place* or *different time*. The reason for this is that the groupware becomes the only available communication channel between the workers. According to P.H.Carstensen and K. Schmidt [8], the groupware have to comprehend the complexity of human communication, including support for awareness, and both verbal and non-verbal communication, as well as formal/informal communication.

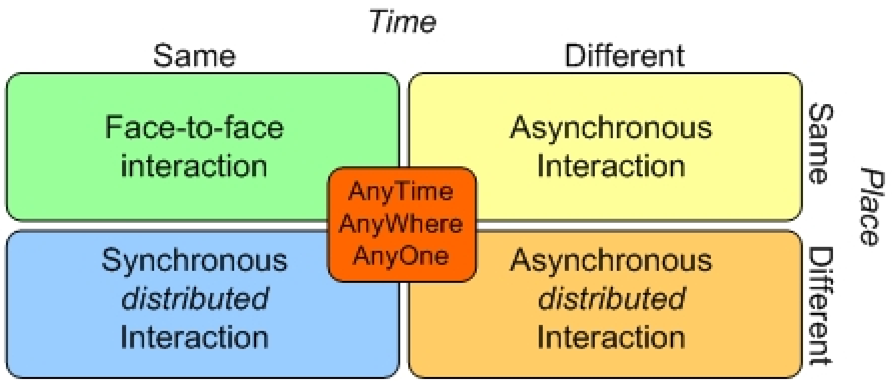


Figure 9.1: *Clasifying groupware with the time-place matrix*

Groupware can further be divided in two groups:

Mobile groupware: The target which Peer2Me focus on, involving mobile phones PDAs. Since Peer2Me uses Bluetooth as means of communication, the framework is designed for applications in the same place, but both real and asynchronous time and space.

Non-mobile groupware: The most common type of groupware. Designed to run on systems that you normally do not carry around while in use, such as desktops and laptops. A laptop might be quite mobile, but the user is seldom moving around while using the groupware software.

Since Peer2Me uses the principle of PAN, peers needs to be in close reach of each other, such as face-to-face, either planned or unplanned, formal or informal. One can consider the coverage area of a peer as a sphere, and when two of these spheres comes within reach of each other, collaborative applications can be run. Figure 9.2 illustrates the principle of spheres.

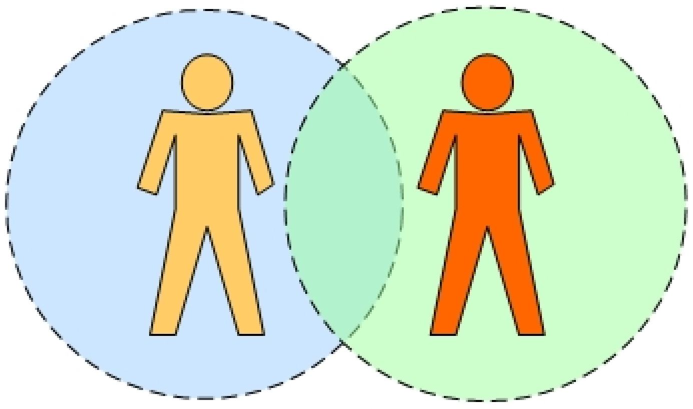


Figure 9.2: *Digital spheres intersecting*

Typical collaborative P2P applications using Peer2Me can have functions such as:

- Exchange personal data
- Spread rumours

- Alert the user of friends that are nearby in high-population places
- Match making
- Identify people we want to meet
- Trade delivery tasks

Many of the examples mentioned above are typical examples of impromptu collaboration, and suited for mobile collaboration, even though several of the examples above will also be valid in a non-mobile environment. Comparing mobile and non-mobile collaboration reveals that most of the collaboration-software running on a non-mobile system, will be able to run on a mobile system if the hardware requirements are met. Mobile collaboration software will often provide more value and greater flexibility for the users.

CHAPTER 10

State of the Art

Peer2Me is developed using new and popular technology, and there are several projects similar or related to Peer2Me. This chapter outlines some of these projects and clarify potential similarities.

Most of these projects are either open-source or research projects. This gives an indication of how immature the technology is; the technology is not yet ready for commercial release nor release to the public.

10.1 Bluecove

Bluecove is an open source implementation of the JSR-82 Bluetooth API for Java. Initially, Bluecove only supports the Windows XP SP2 Bluetooth stack, but will support other operating systems and other Windows stacks in later versions. The Bluecove library can be used to connect a PC with a Bluetooth dongle, and a mobile phone with JSR82 support, together in a Java environment.

The Bluecove library can be downloaded from SourceForge [14], and Ben Hui has created a guide which explains how to take usage of Bluecove. This can be found at the webpage of Ben Hui [18].
We have tested Bluecove using this guide with positive results.

While Peer2Me is meant for ad hoc connection between mobile phones, Bluecove is designed to connect a mobile phone and a computer. Bluecove does not allow connection between two mobile phones. Using the Peer2Me framework together with Bluecove, one can create a robust framework that can connect several computers and mobile phones together, but the two libraries may overlap each other. Peer2Me and Bluecove will *not* focus on the same usage area.

10.2 Umbrella.Net

Umbrella.Net is an experimental platform for developing ad hoc networks based around coincidence or chance occurrences. The project utilizes the unpredictable patterns of weather and crowd formation as a base for network formation. The system consists of a set of umbrellas functioning as nodes. Umbrella.net establishes a visual footprint of a network in public space and creates a framework for sharing localized information among connected nodes. A handheld device, typically a PDA, is connected to a real umbrella, starting a service discovery and connecting to other people with their umbrellas open. People can now share information ad hoc. This information can be anything from chatting to a sudden market were people can buy and sell from each other, and nodes do not need to be in direct connection with each other, which means the information can pass through many nodes before reaching the destination node. This principle is mentioned in Section 8.5, called a *scatternet*, using multihoping. A simple graphical description is shown in Figure 10.1 below.

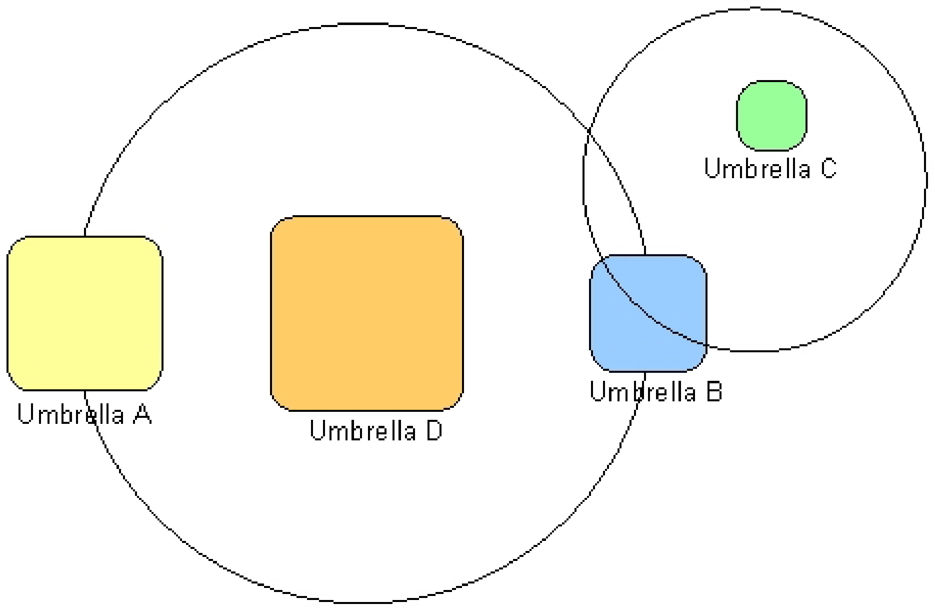
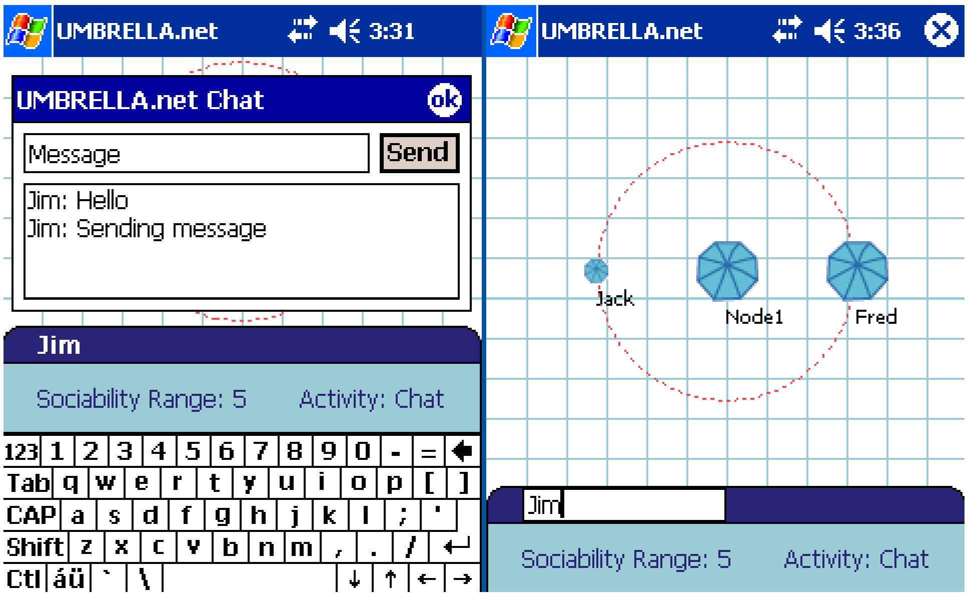


Figure 10.1: *Typical umbrella.net schema*

Umbrella.Net is developed to run on a PDAs operating Windows CE only. Example screen shots of the Umbrella.Net GUI is shown in Figure 10.2.


Jim: Hello
Jim: Sending message

Sociability Range: 5 Activity: Chat

Figure 10.2: *The GUI of the Umbrella.Net software*

Umbrella.Net seems to have the same goals as Peer2Me, except Umbrella.Net's target platform is a PDA running Windows CE.
There are no official details regarding programming platform or architecture. Though the last news available at Umbrella-Net's homepage is dated September 2004, the project has received high recognition and been exhibited in several countries like Ireland, Switzerland, USA and Australia.

Umbrella.Net and Peer2Me can not be seen as competitors, since Umbrella.Net is a complete software package designed for end-users, while Peer2Me is a framework which developers can take advantage of. There are currently no release date for Umbrella.Net.

10.3 BEDD

BEDD is a software suite developed and maintained by the BEDD Corporation [4] that enables ad hoc mobile communication between supported mobile phones, which is any mobile phones using Symbian Series 60 and supporting Bluetooth. BEDD runs in the background on your mobile phone, and automatically searches for other "BEDD phones" near by. When BEDD finds new units running BEDD, it automatically exchanges your profile about whom you are and who you would like to meet. It also exchanges ads about things that you would like to buy or sell.

The software can be downloaded for free from BEDD's website, and is also free to distribute. Among the applications in the suite we find:

BEDDmates - Matches your profile against others and alerts the user when a match is found.

BEDDbay - Marketplace. Place ads and view other ads.

BEDDtalk - A chat application

BEDDfish - Allows the user to send free text messages to other user that does not have BEDD installed and running

BEDDings - Sounds, alerts, themes etc.

BEDDbios - Your profile for others to see

BEDDbuddies - Get alerted when friends are within range

BEDDshare - Allows the user to share the BEDD software

Since our last review of BEDD in our depthstudy [5], BEDD has signed a deal with Nokia for pre-installation of the BEDD software in the Nokia 6600, 6670 and 7610 mobile phones.

As Lund and Norum stated in their master thesis [31], BEDD is closely related to Peer2Me. The difference is that BEDD is an end user application and not an open development platform. The applications available in the BEDD software suite is exactly the kind of applications that can be developed using J2ME and Peer2Me. There are no official news regarding the future of BEDD, but one must assume that it will evolve and gain popularity.

10.4 Other projects

The following projects are mentioned by Lund and Norum in [31] and we will quickly review them.

10.4.1 JXTA

JXTA, Juxtapose, is a language- and platform-independent open-source platform for peer-to-peer networking, developed by Sun Microsystems. This platform enables developers to create distributed computing software for connecting several devices together in an ad hoc fashion. The JXTA is itself to heavyweight to be deployed on mobile phones, so Sun created JXME for this purpose. JXME is a limited version of JXTA intended to run on a CLDC based device (J2ME).

For more information on JXTA, please read "JXTA: A Technology Facilitating Mobile Peer-To-Peer Networks" by Nico Maibaum and Thomas Mundt [33].

10.4.2 JSR82 259

JSR82 259, referred to as the **Ad Hoc Networking API**, is a library for J2ME that enables ad hoc communication between mobile devices in a peer-to-peer environment. The JSR82 API may include methods for:

- Service discovery
- Service registration
- Service availability alert
- Service and service capability inquiry
- Remote service consumption

We mentioned in our depthstudy [5] that the final approval ballot was set to the end of 2005. It has been voted upon and approved by companies such as IBM, Motorola, Siemens, Nokia etc.

JSR82 259 will offer much of the same functionality as Peer2Me, but will *probably not* include messaging functionality.

10.4.3 OBEX

OBEX, the Object EXchange protocol, is maintained by the Infrared Data Association and is designed to allow exchange of binary dataobjects over an infrared link. It has also been adopted by Bluetooth SIG and SyncML wing of the Open Mobile Alliance (OMA). OBEX has more or less become the standard used when send binary data directly between mobile phones, computers, PDAs etc.

There are several ongoing projects using OBEX, among these the open source project **Open-OBEX**, carried out at the University of Tromsø. OBEX is further described in Section 11.3.1. For more information regarding the Open-Obex project, please visit the project homepage [38].

10.4.4 SyncML

SyncML, *Synchronization Markup Language*, an industry-wide effort to create a single, common data synchronization protocol optimized for wireless networks. SyncML is intended to work on transport protocols as diverse as HTTP, WSP (part of WAP) and OBEX, and with data formats ranging from personal data (e.g. vCard vCalendar) to relational data and XML documents. The SyncML consortium was set up by IBM, Nokia and Psion among others. SyncML is mostly used to synchronize contact and calendar information betweena handheld device and a computer. Current version of SyncML is 1.1.2. More information on SyncML can be found on *Wikipedia*, see [46] and [43].

10.5 Summary

All the technologies and solutions mentioned in the previous sections relate to mobile P2P networking, but comparing them to Peer2Me reveals differences.
Peer2Me and BEDD is designed to work on mobile phones, while Bluecove should run on a desktop computer and Umbrella .NET is designed to run on a PDA. These software solutions will manage to do some of the same tasks, but is designed to run on different platforms.

The projects most comparable to Peer2Me are JXME and JSR259. JXME is as mentioned a limited version of *JXTA*, and as Peer2Me it can be used by developers to easily create the P2P-model. JSR259, developed by Sun Microsystems Inc., is the newest project among the mentioned, and offers must of the same functionality as Peer2Me. These two projects are the nearest competitors, except the Peer2Me framework offers more functionality.
Looking at BEDD, it is a complete off-the-shelf solution, unlike Peer2Me, it is a P2P software solution offered to end-users. BEDD can be downloaded and used directly. One could create a solution like BEDD by using Peer2Me. Other projects mentioned in Section 10.4, *OBEX* and *SyncML*, are protocols allowing a standardized way of sending data. OBEX are as mentioned used to send data directly between mobile units, and could be incorporated into an existing communication-framework, like Peer2Me or JSR259.

The conclusion must be that compared to the projects mentioned above, Peer2Me is unique when it comes to functionality and value.

CHAPTER 11

Technology

This chapter provides an update on the latest technology related to mobile ad-hoc networking, and is partially based on the research performed in Chapter 9 in our depthstudy [5]. Since this project involves several mobile hardware and software technologies, it is important to see if new technology or updates in existing technology can affect the framework. This is mentioned in the research questions in Chapter 4. There is also a brief overview on how some of the technologies handles security. Security can be essential when developing software based on a certain technology.

11.1 Mobile Phones

Mobile phones are still the main deployment platform for Peer2Me, and there are no perceptible technology updates regarding mobile phones. Java Micro Edition 2 (J2ME) and Bluetooth is required, and a sensible screen resolution of minimum 130x170 pixels is recommended. Peer2Me can also be deployed on PDAs and other units meeting these requirements. If a device can run the application in the background while doing other tasks, will depend on the device. As an example, a Sony Ericsson W800i can minimize a java application while performing other tasks like answering a phone call etc., and maximize the application afterwards. This can also be done when running the Peer2Me framework.
During this project we are mainly using Sony Ericsson phones to test the software on:

Sony Ericsson K750i A popular model supporting J2ME MIDP2.0 and Bluetooth, and a screen resolution of 176x220 pixels.

Sony Ericsson W800i The same specifications as K750i, except a different casing and some modifications in the software focusing on mediaplaying.

During our depthstudy [5], we tried to use both a Siemens S65 and a Sony Ericsson P900, but experienced several problems regarding threading and other problems as mentioned in our depthstudy, Chapter 9 [5].

11.2 Java 2 Micro Edition

A mobile phone can not be compared to a desktop device due to limited resources, and it is not possible to use desktop software on the mobile phone. Therefore, Sun developed Java 2 Micro Edition (J2ME) because Java 2 Enterprise Edition (J2EE) nor Java 2 Standard Edition (J2SE) could be used. Doing this, SUN made it possible to run Java applications on consumer and embedded devices as well.

11.2.1 J2ME architecture

The Java architecture defines elements for building a complete Java runtime environment that meets the requirements of a wide range of devices. The elements consists of profiles, configurations and optional packages that are structured hierarchically as shown in Figure 11.1.

J2ME is a part of the Java technology, and is, as mentioned, targeted for mobile devices. This means there are strict limitations on memory, processing-power and I/O-capabilities. J2ME is designed to operate within these restrictions. To implement in J2ME, a configuration composed of a virtual machine and a minimal set of class libraries are needed. There are currently two configurations available for J2ME:

CDC - Connected Device Configuration. Used generally by devices that have more resources available such as PDAs and set-top boxes.

CLDC - Connected Limited Device Configuration. Designed for mobile phones, and are available in two versions, 1.0 (JSR82 30) and 1.1 (JSR82 139). 1.1 is a revised and backwards compatible version of 1.0. Version 1.0 will be used in this project.

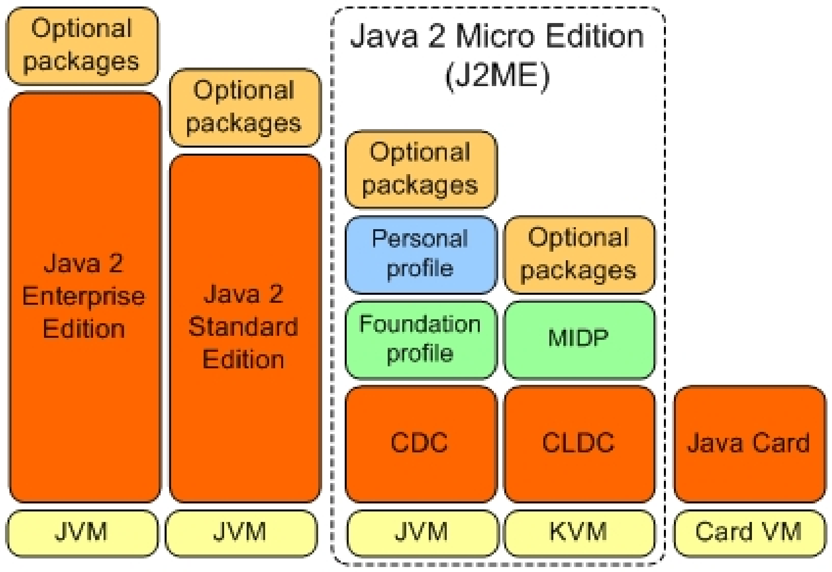


Figure 11.1: *Overview of the Java enviroments*

To define the life cycle, user interface and access a device's properties, a *profile* is needed. Sun has defined the profile, Mobile Information Device Profile, MIDP, for this purpose. MIDP defines a set of available functionality, and can be extended with optional packages. Some of the optional packages will not function, even if the mobile phone support J2ME and the current MIDP, which is version 2.0. For more information, read the J2ME whitepaper [21] and see Sun's website [24].

11.2.2 Optional packages

As mentioned above, several optional packages are available for J2ME, and the following packages are used in this project.

JSR82 75 - The Java APIs for accessing PIM (Personal Information Management) data and the filesystem. This package is aimed at PDAs and high-end mobile phones.
By having access to a device's PIM-information, an application can extract information from the phones calendar, address book etc. And having access to the filesystem is very useful and gives the opportunity to create powerful applications. One of the applications developed in our depthstudy, PeerShare (see Part V in the depthstudy [5]), used the JSR82 75 package.

JSR82 82 - The Java APIs for Bluetooth. This is a specification that standardizes a set of JAVA APIs to allow Java-enabled devices to integrate into a Bluetooth environment. The Peer2Me framework depends on this package and must be imported in the J2ME MIDlets that imports Peer2Me.

11.3 Wireless network technologies

Several different network technologies have been developed and are targeting different usage areas offering divers functionality and specification. This chapter will outline the most common technologies, and provide a brief update on new technology and upgrades on the existing.

11.3.1 Bluetooth

Bluetooth is a common wireless technology mainly used for connecting and exchanging information between devices like mobile phones, PDAs, computers etc. It is an industrial specification for wireless personal networks (WPANs) and operates via a low cost, short range radio frequency (2,4GHz). Bluetooth development is controlled by The Bluetooth Special Interest Group (SIG) [40].

Bluetooth exists in several versions which defines different speed, range and power consumption. The range and power consumption are divided into three classes:

Class 1: Range up to 100m and a 100 mW power consumption

Class 2: Range around 10m and a power consumption of 2.5mW

Class 3: Range less than 1m and a power consumption of 1 mW

As shown above, there is a correlation between range and power consumption. Power consumption is essential when integrating Bluetooth into mobile units, but then again it is a tradeoff versus range. The most common combination used is class 2. This gives adequate range while keeping the power consumption at a reasonable level. A hybrid between class 1 and 2 is often used in mobile phones.

Bluetooth did first arise as version 1.0, but is commonly known as version 1.1 or 1.2 which is most widely implemented. 2.0 is the latest version, and the greatest improvement is the introduction of *Enhanced Data Rate* which increases the speed to a maximum of 2.1Mbit/s. About three times faster than version 1.1 and 1.2. Version 2.0 has also an improved support for scatternet and shorter discoverytime.
Bluetooth also supports a set of *profiles*, each a description of the Bluetooth unit; a Bluetooth hands free would use the *Headset*-profile, while a Bluetooth mouse would use the *HID*-profile, *Human Interface Device*, and so on.

Bluetooth Services

In order to connect two or more Bluetooth devices, at least one must run a *service* and one must be *discoverable* for other devices. The device that initiates the connection to another device, becomes the master, while the device that runs the service becomes the slave. Many devices have the capability of switching between a master and a slave configuration.

Bluetooth Communication

The Bluetooth specification contains a Bluetooth protocol stack which is responsible for all data communication. Bluetooth devices can use several different communication protocols in order to communicate with each other. The Java API for Bluetooth allows data communications for three protocols: L2CAP, RFCOMM and OBEX [32].

L2CAP : The underlying protocol which all Bluetooth communication is layered upon. It multiplexes data between different higher layer protocols and provides QOS, *Quality Of Service* management for higher layer protocols.

RFCOMM : The RFCOMM protocol emulates an RS-232 serial connection which provides a stream-based interface to the RFCOMM protocol. Important aspects regarding RFCOMM are:

- Only one RFCOMM session at a time can be shared between two devices.
- The maximum amount of active RFCOMM services a Bluetooth device can have is 30.
- A Bluetooth device can only support one client connection to a service at a time.

The RFCOMM protocol is very straightforward to implement in J2ME. However, the protocol has limited functionality and does not scale very well for more advanced applications. The only option the developer has is to write to an outputstream and close it. Then the receiving Bluetooth device must read the stream and parse it. The old version of Peer2Me used the RFCOMM protocol for all purposes, including message exchange. In our depthstudy [5], we found severe weaknesses with their implementation regarding messages. The implementation only allowed for simple text strings to be sent over the protocol. When an application receives a message, it very often wants to know who sent it, the length of the message, at which time it was created, if it were sent to other devices as well, etc. In order to achieve this, the old version of Peer2Me created message headers which where constructed as shown in Listing 11.1.

Listing 11.1: *Message header from the old Peer2Me framework, from* `messageHeader.txt`

```
from: no.ntnu.idi.mowahs.project.bluetooth.domain.LocalBluetoothNode:0000F5EB5AD5;B;
to: no.ntnu.idi.mowahs.project.bluetooth.domain.RemoteBluetoothNode:0123456789AF;A;
to: no.ntnu.idi.mowahs.project.bluetooth.domain.LocalBluetoothNode:0000F5EB5AD5;B;
msgtype: 1
msgbody:
txt.pan_im_message: HELLO
txt.service: peer2me_pan_im
```

The old version did actually not support the use of a linebreak in a message string since it searched for linebreaks in order to parse the headings in the message. A very good alternative to RFCOMM is OBEX where this problem does not occur at all.

OBEX : In contrast to HTTP, OBEX (Object Exchange Protocol) is stateless. OBEX is much more versatile than the RFCOMM. In order for two devices to communicate, an OBEX session has to be created. The device that initiates the creation of a session is the *client*, and the other device is the *server*. Then, the client can send header information to the server, followed by a put or get request. The Java implementation of OBEX has 12 predefined headers and allows the developer to create an additional 64 headers as illustrated in the API for javax.obex [41]. This gives the developer freedom to attach all sorts of information to a specific message. Using OBEX as the transport protocol in the new version of Peer2Me, will make it easier to incorporate advanced messaging in the framework.

The OBEX protocol defines eight operations as explained on the Sony Ericsson developer site [11]:

- CONNECT: Sets up connection and creates session

- PUT: Sends an object from the client to the server
- GET: Requests an object from the server and receives the object
- SETPATH: Changes active directory on the server
- CREATE-EMPTY: Creates an empty object on the server
- DELETE: Removes an object from the server
- ABORT: Terminates an operation such as PUT before the operation has completed
- DISCONNECT: Ends an OBEX session

The Peer2Me framework does not necessarily need to implement all these operations. The most important operations such as: CONNECT, PUT and DISCONNECT must however be implemented. The GET operation can easily be implemented by using two PUT operations. For example, if a client wishes to download a file from a server, it can simply send one GET message and receive the file in the same operation if the server accepts. It can also send one PUT message which contains information the server can interpret as a request to download a file. The server can then use the PUT command on the client in return.

It is possible for a Bluetooth enabled J2ME application to access a service that uses another protocol such as TCP/IP, but the protocol must then be implemented in the application using the CLDC Generic Connection Framework.

Bluetooth Security

The security can be decided by the users and are divided into three levels:

Level 1: No security functionality. Connection is being made without any encryption or authentication.

Level 2: Service level security. Security measures like access control to devices and services are activated to control which units can use different services on other units.

Level 3: Link level security. At this level security is based on a common shared key generated during the pairing process. This process also involves a PIN-code entry from the users.

Even though Bluetooth implements the security-measures above, it does not provide end-to-end security, and is therefore exposed to several security issues discussed among others in a document published by Adam and ben Laurie in the article *Serious flaws in Bluetooth security lead to disclosure of personal data* [28].

Bluetooth Updates

The Bluetooth SIG (Special Interest Group), which maintains the Bluetooth standard, has recently joined forces with the WiMedia Alliance to further develop the Bluetooth technology [34]. They plan to develop a UWB (Ultra Wide Band) chip that supports high speed data transfer, yet still consumes low power. The new chip will also be backwards compatible.
In a recent press release, The Bluetooth SIG announced:

> "It is critical that the UWB technology be compatible with Bluetooth radios and maintain the core attributes of Bluetooth wireless technology low power, low cost, ad-hoc networking, built-in security features, and ability to integrate into mobile devices. Backwards compatibility with the over 500 million Bluetooth devices currently on the market is also an important consideration."[34]
>
> Bluetooth SIG estimates that the first Bluetooth technology/UWB solution chip sets will be available for prototyping in the second quarter of 2007.

11.3.2 Wireless Local Area Network

WLAN, short for Wireless Local Area Network, is commonly known as a wireless technology that is intended to allow mobile devices to communicate, get Internet access or connect to local area networks. It is also used for wireless Voice over IP phones (VoIP).
WLAN is one of the most common used wireless technologies and is divided into four main standards with different operating frequencies and bandwidth, where 802.11g operating at maximum of 54 Mbit/s is the most common among end-users. Lately, the focus is shifting to the new upcoming standard of wireless local area network, 802.11**n**.

IEEE 802.11n

The *n*-standard is being drafted by large communication-corporations as Broadcom, Intel and Philips, and final standardization-proposal are expected at the end of 2006. The performance 802.11n will exceed every present standard and is expected to be 10 times faster than the current 802.11g-standard, delivering speeds up to 540 Mbit/s.
There are several producers of network equipment that are delivering devices based on a preliminary draft, labeling their equipment with *Pre-N.* Pre-N products utilizes parts of the technology in 802.11n such as *MIM0* and *OFDM*, explained below:

MIMO - Multiple-input multiple-output, is an abstract mathematical model for some communications systems. In radio communications if multiple antennas are in use, the MIMO model naturally arises because the signal would choose different paths and arrive asynchronously. Normally this is a problem, but the MIMO-technology takes advantage of this and can send more data at the same time than the technology used in previous 802.11 versions.

OFDM - Orthogonal Frequency-Division Multiplexing, is a technique using a single transmitter to transmit on many different independent frequencies, typically dozens to thousands, and because the frequencies are so closely spaced, each has only room for a narrowband signal. This modulation technique coupled with the use of other advanced modulation techniques, results in a signal with high resistance to interference.

For more information on MIMO or OFDM, please read *Using MIMO-OFDM Technology to Boost Wireless LAN Performance Today* by Datacomm Research Company [9].

Wireless LAN transmissions can be either *open* or *secure.* The security is implemented by encrypting the transmission using different techniques and methods. The most commonly used is *WEP*-Wireless Equivalent Privacy, which uses a shared key and encodes the data using either a 40bit, 104bit or a 232bit encryption key (often referred to as 64, 128 or 256bit encryption). Uses RCA encrypting and TKIP. Another upcoming encryption- method is the *WPA/WPA2* - Wi-Fi Protected Access, which uses a passphrase as key. This encryption method avoids many of the security-flaws which occurs in the WEP-encryption. WPA2 requires newer equipment. For more information on issues regarding the security of WEP, please read *Intercepting Mobile Communications: The Insecurity of 802.11* by N. Borisov, I. Goldberg and D. Wagner [37].

11.3.3 ZigBee

ZigBee is a small, low powered digital radio based on the IEEE 802.15.4 specification using high-level protocols for communicating on low speeds within a range of 10-75 metres. ZigBee is designed to be simpler and cheaper than other Wireless PANs, such as Bluetooth, but due to a low market demand, the production cost is still higher than i.e. Bluetooth.
ZigBee, driven by the ZigBee Alliance, are currently working on a certification-program to ensure that ZigBee-products can communicate ”out of the box”.

ZigBee offers a service based, high level protection, using encryption, access control, frame integrity and sequential freshness. The security features of ZigBee is discussed in *Industrial-strength Security for ZigBee: The case for public-key cryptography*, written by Mitch Blaser [6].

11.3.4 Radio Frequency IDentification

Radio Frequency IDentification, RFID, is an automatic identification method, relying on storing and remotely retrieving data using RFID tags and transponders. An RFID tag is a small chip which can contain a limited amount of information, and can be incorporated into a product, animal or person. There are different types of RFID-tags available, both *active* and *passive*. The main difference is that active tags uses a fixed internal power supply, which gives larger storage capacity and great range, whereas passive RFID tags have no internal power source. The passive tags rely on inducing enough power through the radio frequency to transmit a response when called upon. There are also semi-active tags which are quite similar to passive tags, except using a small battery.
The purpose of an RFID system is to enable data to be transmitted by a tag, which is read by an RFID transponder and processed according to the needs of a particular application. The data transmitted by the tag may provide identification or location information, or specifics about the item tagged, such as price, color etc.

RFID has no implemented security measures, and introduces great challenges regarding personal information security and access security. There are no well known attempts at designing a good system which handles security issues in RFID.

11.3.5 Wireless USB

Wireless USB, WUSB, is a new wireless extension to USB intended to combine the speed and security of wired technology with the ease-of-use of wireless technology. WUSB is based on ultra wideband wireless technology defined by IEEE 802.15.3a (yet to be accepted), which operates in the range of 3.1-10.6 GHz. WUSB offers bandwidths of 480 Mbit/s at three meters and 110 Mbit/s at 10 meters. The maximum bandwidth of 480 Mbit/s is the same as the bandwidth of USB2.0.
WUSB uses a point-to-point hub-and-spoke topology as shown in Figure 11.2. With wired USB, in opposite to wireless USB, no hubs are present in the connection topology.

With the growing use of digital media in the PC, consumer electronics and mobile communication environments, a common standard is needed to support the on-going convergence of these environments. The trend toward simple and convenient wireless distribution of digital information gives an opportunity to introduce a single, standard wireless connection capable of supporting the needs of most technological environments.

WUSB security will ensure the same level of security as wired USB. Connection-level security between devices will ensure that the appropriate device is associated and authenticated before operation of the device is permitted. Security measures as encryption must be implemented in the software using wireless USB.
The security features in WUSB will not affect neither performance nor production costs.

As the wireless USB specification is not officially released, products using this technology will not appear in a large number until a proper release and certification program has taken place.

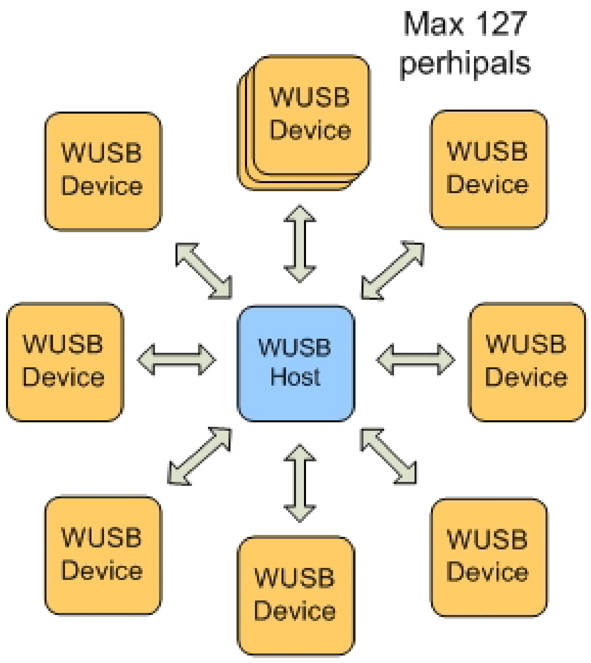


Figure 11.2: *The Wireless USB topology*

11.3.6 Wireless Firewire

Wireless firewire is a new wireless extension to firewire based on the IEEE 1394 over IEEE 802.15.3 standards. When the new wireless firewire is released, it will retain the speed of the wired version (up to 400 Mbit/s), and a theoretically up to 200 wireless firewire-units can be in the same WPAN. The main usage area of wireless firewire would be within media, hence transmitting video and sound to/from set-top boxes, from video cameras etc. Much the same as the wired version of firewire.

The security of firewire would be much like the principle used in wireless USB described in Section 11.3.5, the same as the wired version. As for wireless USB, there are no official products using wireless firewire on the market.

CHAPTER 12

Software Architecture

Software architecture is a field of study that is of high importance to the Peer2Me framework. It is possible to make such a framework without paying any attention to the architecture, but doing so will result in a poor, low-quality, non-flexible and over-complex framework. Software architecture can be used to ensure that the system meets the quality requirements set out by the different stakeholders of a system. This chapter contains information regarding quality attributes associated with an architecture, the involved stakeholders, architectural tactics[1] and architectural patterns[2] that can be used to achieve quality requirements. This is explained by Bass, Clements and Kazman in *Software Architecture in Practice* [3]. Much of the material in the preceeding sections are from this book.

12.1 Stakeholders

In every development project there are stakeholders. The stakeholders are people who in some way have interest and/or influence in the project. Their interests have great impact on the quality of the system. Often, it is nearly impossible to satisfy all the requests. If the maintainer wants the system to be highly modifiable and flexible, and the marketing manager wants it to have short time to market, a conflict arises. The architect has to choose either high modifiability or short time to market.
Typical stakeholders in a project are: Developers, architects, testers, maintainers, end-user, funders, project manager etc. In the Peer2Me project we can define the following stakeholders: developers, maintainers, architects, end-users, testers and supervisor. The original architects of the Peer2Me project are Michael Sars Norum and Carl Henrik Wolf Lund. Now, the authors of this report, Tommy Bjørnsgård and Kim Petter Saxlund are the developers and architects. The end-users are developers who we hope will adopt the framework when it is ready to be released. The maintainers are possible future master students. The testers can be us or some other independent developers that might benefit from the Peer2Me framework. Our supervisor, Alf Inge Wang, is also a stakeholder. His primary focus is on usability and modifiability, since he will probably guide other students in future work with Peer2Me.

Since we are not going to continue work with Peer2Me after we have developed it, we must

[1]Architectural tactics: Known methods for achieving quality in a system

[2]Architectural patters: A description of element and relation types together with a set of constraints on how they may be used

make sure that our followers can easily adopt the framework and understand why it has be designed the way it has. It also has to be easy for other developers to just use the framework in their applications. Based on this, a set of quality attribute scenarios for the Peer2Me framework will be presented in Part IV. Quality attribute scenarios are presented in Section 12.2.

12.2 Quality of a System

Functionality is most often the center of attention in a development process. If quality attributes are ignored, the result is most often a poor and unmanageable architecture. A quality attribute does not necessarily entirely affect the architecture. For instance, a usability attribute may be the background color of an application window which has no impact on the architecture itself. However, giving the user the ability to change the background color has impact on the architecture.

12.2.1 Quality Scenarios

There are several ways of defining non-functional requirements for a system. One solution is to use quality attribute scenarios. These scenarios describes the quality requirement in detail, which is very useful when designing the architecture. Such scenarios consists of six parts:

Source of stimulus: The entity that generates the stimulus. Can be either human or computer system.

Stimulus: The condition that arrives at the system and needs to be considered.

Environment: The stimulus occurs in a certain environment. Examples of environments are: runtime, design time, testing phase, etc.

Artifact: The artifacts that is stimulated. An artifact can be pieces of a system or the whole system.

Response: When the stimulus arrives at the system, the system has to respond in some way.

Response measure: The response should be measurable so that the attribute scenario can be tested.

We have two types of scenarios: General and specific. Table 12.1 shows a general availability scenario. The general availability scenario can be used to derive to specific scenarios. A specific availability scenario is presented in Table 12.2. Availability is only one out of many quality attributes that can be defined. The attributes are described in Section 12.2.2.

General availability scenario	
Source of stimulus	Internal, external
Stimulus	(Fault) Omission, crash, timing, response
Environment	Normal, degraded operation
Artefact	Process, storage, processor, communication
Response	Record, notify, disable, continue (normal/degraded), be unavailable
Response Measure	Repair time, availability, available/degraded, time interval

Table 12.1: *General availability scenario [3]*

Sample availability scenario	
Source of stimulus	External to system
Stimulus	Unanticipated message
Environment	Normal operation (Runtime)
Artefact	Process
Response	Inform operator and continue to operate
Response Measure	No downtime

Table 12.2: *Sample availability scenario [3]*

12.2.2 Quality Attributes

Non-functional requirements can be divided into several categories. The categories are not completely isolated from each other. If the system must be highly available, then high performance will be hard to achieve. The categories are:

Usability: Usability is concerned with how easy it is for a user to perform a certain task and how the system displays information to the user. Usability is an issue that often must be considered in the early stages of architectural design. If major problems regarded to usability is detected late in the project phase, the more repair and modification has to be done to the architecture.

Modifiability: Modifiability has to do with changes to the system. It is then vital that the changes can be performed without much hassel. For instance, if a maintainer wants to change an encryption algorithm, then he/she should only need to replace one module of the system and not many small changes in many modules. A change does not necessarily need to be made by a maintainer/developer. It can also be made by the end-user, for instance in a configuration set-up.

Performance: Basically, performance has to do with how long the system can respond to an event that occurs. Such events may come from several instances. These instances can be an end-user, the system itself or from other systems. For example, a user might try to login to an ftp-server and how long it takes for the server to log the user in will be a performance scenario.

Availability: A system's faults and failures are associated with availability. A fault occurs when something goes wrong in the system and is not visible. If a fault becomes a failure, the error will be visible. For instance, if the network connection is lost between two devices without one device registering the loss, a fault has occurred. Then, if the application acts as if the connection is still available, a failure will occur.

Security: Security is concerned with the system's ability to prevent unauthorized usage/access without compromising normal usage. Attacks can be unauthorized attempts to access or modify data.

Testability: In order to find bugs and faults in the system, it needs to be testable. Designing an architecture that can be easily tested for faults will save a lot of time. There are several different ways of doing this. Most of them involve monitoring the system's internal state and logging output that is easy to interpret.

12.3 Architectural Tactics

In order to achieve the different qualities, the architect has to use certain architectural tactics. Architectural tactics are known methods for achieving quality in a system. There are many known tactics that can be applied to know issues. Often, many tactics are used to achieve a certain quality, this is called architectural strategy. Bass, Clements and Kazman [3] have defined several tactics that can be used to fulfill all the six quality attributes mentioned in Section 12.2.2. These are described in the following sections.

12.3.1 Modifiability Tactics

The Peer2Me framework should be easy to maintain and it is therefore necessary to include modifiability tactics in the architecture. Modifiability tactics can be divided into three categories: localize modifications, prevent ripple effects and defer binding time.

Localize modifications: The following tactics will ensure that modifications are localized in the future:

- *Maintain semantic coherence* - Assigning modules specific responsibilities and make them work together. Modules should not be to dependent on each other. It is also important to extract common services. By doing this, future changes to these services will only have to be done at one place.
- *Anticipate expected changes* - This tactic has much in common with the previous tactic. This tactic is concerned with minimizing the effects of the changes. It is difficult to use this tactic alone, and it is therefore used in conjunction with maintaining semantic coherence.
- *Generalize the module* - Making a module more general will allow it to handle a broad range of functions based on input. The input type can then vary between a broad range of types. In the future, the changes will then only affect the input type.
- *Limit possible options* - By limiting the options of future changes, modifications can be localized. For instance by allowing the change of a CPU, but restricting the change to one product family, the options will be limited.

Prevent ripple effects: A ripple effect is when a module needs to be changed just because another module has been changed. This often occurs when the dependency between two modules are too high. The following tactics can be used to prevent ripple effects:

- *Hide information* - When making modules, the developer should carefully consider which information should be made public and which should be kept private. Isolating changes to one module will prevent modifications to propagate to other modules.
- *Maintain existing interfaces* - By maintaining existing interfaces, modules that use the interfaces do not need to change. It is often difficult to mask data, so creating general interfaces is often a good idea. There are several patterns that implement this kind of tactic: adding interfaces, adding adapter and providing a stub.
- *Restrict communication paths* - Try to keep the sharing of data to a minimum between the different modules. There is often no need to share the data with several modules.
- *Use an intermediary* - An intermediary takes care of all activities between two modules. There exist several types of intermediaries. For more information about these, see *Software Architecture in Practice* by Bass et. al. [3].

Defer binding time: Changes can be done by non-developers by using some of the following tactics:

- *Runtime registration* - plug-and-play operation at runtime or load time.
- *Configuration files* - set parameters at startup.
- *Polymorphism* - late binding of method calls.
- *Component replacement* - load time binding.
- *Adherence to defined protocols* - runtime binding of independent processes.

12.3.2 Usability Tactics

Usability tactics are mostly concerned with how the system responds to user requests. Since Peer2Me is only a framework, no visual feedback is given directly to the user. However, the framework should be easy to use for other developers and hence the architecture needs to achieve high usability. Usability tactics can be divided into two categories: *runtime tactics* and *design-time tactics*. Runtime tactics has to do with how the system interacts with the user during execution. Design-time is concerned with design of the system by a developer when it is not running.

Runtime: There are three kinds of human-computer interaction: *user initiative*, *system initiative* and *mixed initiative*. A user initiative is when a user makes the system do something like clicking a button. If the system does some logical operation afterwards, it is called a mixed initiative. If the system does something that the user has not requested, it is a system initiative. The architect must design system response when a user wants to perform a task. In the case where the system has to take initiative, it must use some information about the current situation. It is recommended that the information is encapsulated in some model the system can use to predict the behaviour of the system. There are three kinds of models that can be maintained and used:

- *Maintain a model of the task* - A model of the task the user wishes to perform is maintained so that the system can give proper response. For instance if the user wants to login to a webpage, the system can automatically fill out username and password given that the system has stored username and password for that site.
- *Maintain a model of the user* - A model of the user's common behaviour is maintained. For instance, in WinXP the system "remembers" the last programs that the user executed, it then hides programs that the user never/seldom use and show the common ones.
- *Maintain a model of the system* - A model of the system's behaviour is maintained. For instance the time to extract a large compressed file.

Design-time: There is one tactic that makes it easier to change the user interface after it has been evaluated. The tactic is to *separate the user interface from the rest of the application*. There are four architectural patterns that can be used to accomplish this:

- Model-View-Controller
- Presentation-Abstraction-Control
- Seeheim
- Arch/Slinky

12.3.3 Performance Tactics

Peer2Me is going to be used on mobile phones which have limited resources compared to PDAs and desktop computers. Therefore it is important that the framework does not consume all the resources on the mobile phone. When an event/request arrives at a system, the system needs to respond in some fashion. This consumes resources and can block other processes as well. There are three categories of tactics that can deal with this problem: *resource demand*, *resource management* and *resource arbitration*.

Resource demand: The following tactics can be used to cope with high resource demand:

- *Increase computational efficiency* - Improve the algorithms that are responsible for solving a task.
- *Reduce computational overhead* - Often it can be smart to remove intermediaries to improve latency.
- *Manage event rate* - Reduce the sampling rate at which environmental variables are monitored. Often the sampling rate can be too high which demands a lot of resources.

- *Control frequency of sampling* - If the events of the external generated events are out of control, sampling the queued request at a lower frequency will ease the pressure on resources.
- *Bound execution times* - Set a limit on the time that a can be used to handle a request.
- *Bound queue sizes* - The maximum number of queued arrivals can be controlled which will reduce amount of resources required to handle the arrivals.

Resource management: It is hard to control the demand for resources, but the management of resources is easier to control:

- *Introduce concurrency* - Handle multiple requests in parallel in different threads.
- *Maintain multiple copies of either data or computations* - Caching data reduces contention and makes it effortlessly to produce the data upon request.
- *Increase available resources* - Acquiring faster processors, more memory, more disk space, better networks etc., will increase the available resources and reduce latency. Obviously, there is a significant cost associated to this tactic.

Resource arbitration: The most common scheduling tactics are:

- *First in first out (FIFO)* - All requests are treated equally. The first request will be handled first. A problem occurs when one request takes a long time and other requests have to wait.
- *Fixed-priority scheduling* - By giving each request a priority, the most important requests will be first handled. There are three common strategies for assigning priority:
  - Semantic importance: The priority assigned is associated with some domain characteristics of the task that generates the request. In mainframe systems, this tactic is often used where the characteristic is the time of task initiation.
  - Deadline monotonic - Highest priority is given to the request that has the shortest deadline.
  - Rate monotonic - Higher priority is given to the request that takes shorter time to handle.
- *Dynamic priority scheduling* - There are two type of dynamic scheduling:
  - Round robin - Available resources are assigned to the queued requests in order. This order is often cyclic.
  - Earliest deadline first - The request with the earliest deadline is assigned the highest priority.
- *Static scheduling* - The assignment of resources to requests are determined offline.

12.3.4 Availability Tactics

Peer2Me is dependant on wireless network technology in order to communicate with other devices. Very often, devices that previously were in range becomes out of range. These issues are important to deal with and therefore, availability tactics is of high relevance to the architecture. Availability tactics can be divided in three categories: *fault detection*, *fault recovery* and *fault prevention*.

Fault detection: There are three common tactics for detecting fault in a system:

- *Ping/echo* - In order to check if one module or component is still available, one component can send a "ping" to the other and receive an "echo" in return that confirms that it is still alive.
- *Heartbeat* - One component regularly sends out a heartbeat which another components listens to. If the heartbeat stops then the component is assumed failed and proper actions should be undertaken by the component that listens to the heartbeat.

- *Exceptions* - It is also possible to throw exceptions when a fault occurs. The exception handler is then often located in the same module where the exception is thrown.

Fault recovery: When a fault has occurred, the system needs to recover and return to normal operation. There are several known fault recovery tactics, but most of them are to be used on computers, servers and distributed networks. If a fault occurs in an application that uses Peer2Me, it usually needs to be restarted. This process only takes a couple of seconds, but important information should not be lost when the fault occurs. Therefore, a tactic known as checkpoint/rollback can be used:

- *Checkpoint/rollback* - If a system faults in an unstable state, the system can roll back to a previous working configuration. The different versions of Microsoft Windows has had this functionality for a long time.

Fault prevention: Instead of detecting and recovering from faults, a system can prevent faults from occurring by using one or several of the following tactics:

- *Transactions* - A transaction contains many steps which all must be executed successfully in order to avoid faults. Transactions are often used when several components accesses the same data and locking of resources are used.
- *Process monitor* - A component can watch the processes and terminate a process when it fails. It can then restart the process in an appropriate state.

12.3.5 Security Tactics

A lot of information is expected to be exchanged between mobile devices that uses the Peer2Me framework. Some of this information may be sensitive, which only the intended recipient should have access to. In these cases, proper security tactics must be used. Security tactics can be divided into three categories: resisting attacks, detecting attacks and recovering from attacks.

Resisting attacks: The following tactics can be used in combination with each other to make a system more resistant to attacks:

- *Authenticate users* - To verify that the user is who he/she claims to be. Can be implemented by using passwords, digital certificates, biometric information, password calculators (as used with Internet banking) etc.
- *Authorize users* - Grant correct access to authenticated users. The system will then need some access control patterns. Users can have specific access rights or be member of a larger group of users.
- *Maintain data confidentiality* - Encrypting data provides extra protection when data is transmitted outside a controlled environment. Data can for instance be encrypted and transferred through a VPN or SSL connection.
- *Maintain integrity* - To verify that the data has not been compromised. Checksums can be used for this purpose.
- *Limit exposure* - The lesser services that are available on each host, the safer the system will be.

Detecting attacks: One commonly known way of detecting attacks is by using an *intrusion detection system*. Such a system monitors network traffic and compares it to a database. If it finds something that is abnormal behaviour it must alert the system and take proper action.

Recovering from attacks: Tactics for recovering from attacks can be divided into two categories: *restoring the system* and *identify the attacker*.

- *Restoration* - This tactic overlaps with the fault recovery tactics which is concerned with restoring the system to normal state. The difference is that special attention is given to administrative data such as passwords, access control lists, domain name services and user profile data.
- *Identification* - By maintaining an audit trail, the systems transactions can be traced in order to find the attacker. Often, this audit trail is the attack target and should be highly protected.

12.3.6 Testability Tactics

A problem we discovered in the evaluation of the Peer2Me framework in our depthstudy [5], was proper ways of testing the applications. Therefore, in the new Peer2Me framework, good methods of testing should be available for the developer. Testability tactics can be divided into two categories: *input/output* and *internal monitoring.*

Input/output: In this category we find two tactics:

- *Record/Playback* - Record information that is used as input to a system in order to playback the same scenario later for testing purposes. By doing this, results can be compared in a fast and easy manner.
- *Specialize access routes/interfaces* - Creating special testing interfaces makes it possible to capture variable values in components through a test harness. This can be done independently from normal operation as well. By creating a hierarchy of test interfaces, testing can be done at any level in the architecture.

Internal monitoring: There is one tactic for internal monitoring called *built-in monitors.* A component can watch states, performance load, capacity, security etc through an interface. It should be easy to switch this feature on and off. Often this technique requires more testing efforts as test must be repeated with the feature switched off.

12.4 Design Patterns

A design pattern is a repeatable solution to a commonly-occurring problem in software design. It is not a finished design that can be transformed directly into code, but a description or template for how to solve a problem that can be used in many different situations. Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved. Design patterns can be classified based on multiple criteria, the most common of which is the basic underlying problem they solve. According to this criterion, design patterns can be classified into various classes, some of which are:

- Behavioural patterns
- Creational patterns
- Structural patterns
- Architectural patterns

The next sections will only focus on the patterns relevant to Peer2Me.

12.5 Behavioural patterns

Behavioral design patterns are design patterns that identify common communication patterns between objects and realize these patterns. By doing so, these patterns increase flexibility in carrying out this communication. It provides structured methods for intercommunication between important object within a software solution. A good example of this is the *observer* pattern described in Section 12.5.1.

12.5.1 Observer pattern

The observer pattern, also referred to the *publish subscribe* pattern is a design pattern used in computer programming to observe the state of an object, or publish the result of an operation. The essence of this pattern is that one or more objects (called observers or listeners) are registered to observe an event which may be raised by the observed object.

This pattern is widely used when an application must communicate with other application and receive messages. The application subscribes on the other application's messages.

12.6 Creational patterns

Creational design patterns are design patterns that deal with object creation mechanisms, trying to create objects in a manner suitable to the situation. The basic form of object creation could result in design problems or added complexity to the design. Creational design patterns solve this problem by somehow controlling this object creation. A good example of this is the *singleton* pattern described in Section 12.6.1.

12.6.1 Singleton Pattern

The singleton design pattern is designed to restrict instantiation of a class to *one* object. This is useful when exactly one object is needed to coordinate actions across the system. Sometimes it is generalized to systems that operate more efficiently when only one or a few objects exist. Before designing a class as a singleton, it is wise to consider whether it would be enough to design a normal class and just use one instance.

The singleton pattern is implemented by creating a class with a method that creates a new instance of the object if one does not exist. If an instance already exists, it simply returns a reference to that object. To make sure that the object cannot be instantiated any other way, the constructor is made either private or protected.
Listing 12.1 shows sample code of the singleton pattern in use.

Listing 12.1: *Sample code showing singleton in use*

```java
private static MySingleton mySingleton = null;

public static synchronized MySingleton getInstance()
{
 if(mySingleton == null)
 {
  mySingleton = new MySingleton();
 }
 return mySingleton;
}

private MySingleton(){}
```

As you can see from the listing above, only one instance of *MySingleton* can be created. This example is quite simple, and one must take caution if singleton is used in multi-threaded classes.

12.7 Structural Patterns

Structural patterns are Design Patterns that ease the design by identifying a simple way to realize relationships between entities, helping in abstraction. The two most common structural patterns are the *decorator* pattern and the *facade* pattern. While the decorator pattern is wrapping a decorator-object around the original object including additional functionality, the facade pattern disguises an object hiding unnecessary exposed methods. The facade pattern is described briefly in Section 12.7.1.

12.7.1 Facade Pattern

A facade is an object that provides a simplified interface to a larger body of code, such as a class library. A facade can make a software library easier to use and understand, since the facade has convenient methods for common tasks. It makes the code that uses the library more readable, and reduces dependencies of outside code on the inner workings of a library. The facade pattern can wrap a poorly designed collection of APIs, creating a single well-designed API. Facades are very common in object-oriented design. For example, the Java standard library contains dozens of classes for parsing font files and rendering text into geometric outlines and ultimately into pixels. However, most Java programmers are unaware of these details, because the library also contains facade classes (Font and Graphics) that offer simple methods for the most common font-related operations.

12.8 Architectural Pattern

An architectural pattern is any pattern concerned with the construction context of a whole system, rather than just some part of a system. Architectural patterns are used to describe the structure of bigger systems where the number of objects is measured in a large quantity, and the diversity and complexity of the system is high. This identifies the need of abstraction. In *Pattern Oriented Software Architecture*, F.Buschmann, R.Meunier, H.Rohnert, P.Sommerlad and M.Stal defines architecture patterns as:

> ”Expressing a fundamental structural organization schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them.” [7]

There are many well-known architectural patterns, each suitable in specific situations. Every architectural pattern include a set *benefits* and *downsides* requiring the designer or developer to carefully consider which pattern to use. Choosing the wrong pattern can be more time-consuming during implementation than using extra time when deciding on the appropriate pattern.

Architecture patterns that are relevant to the redesign of Peer2Me is presented in the following sections.

12.8.1 Layered Architecture Pattern

A large system requires *decomposition.* One way to decompose a system is to segment it into collaborating objects. In large systems an initial rough model might produce hundreds or thousands of potential objects. Additional refactoring typically leads to object groupings that provide related types of services. When these groups are properly segmented, and their interfaces consolidated, the result is a **layered architecture**. Even though this might seem complicated, a layered architecture often gives itself, due to obvious domain responsibilities. In an application, the graphical user interface, GUI, should only focus on how data is presented, not how its created. Therefore, the GUI might be a *layer* lying on top of i.e. a data-layer, which creates the data the GUI-layer needs.
Since a layered architecture is quite simple, it can easily be explained to other team members

and easily demonstrate each object's role in the "big picture". Figure 12.1 shows a simple layout of the layers in a fictitious application.

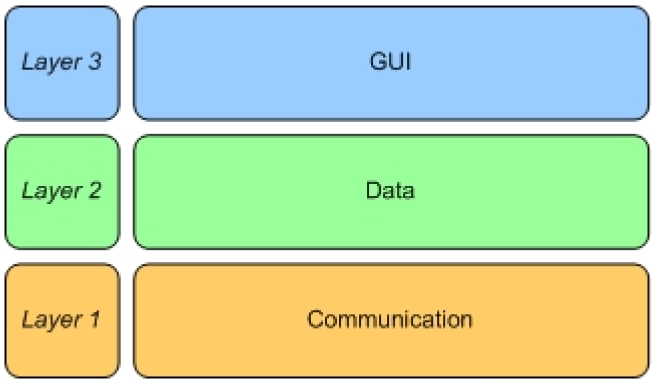


Figure 12.1: *Example of a 3-layer layout in a fictitious application*

A well known example of a layered architecture *put to use* is the 7 layer OSI networking model, assigning each layer its own responsibility. More on this model can be found at Cisco Systems Inc. webpage [20]. It is also worth to mention that several layers can lie parallel on the same "level".

As mentioned earlier, patterns provide benefits and downsides. The layered architecture pattern is no exception. One of the greatest benefits is the easiness of exchanging parts of the software, in other words exchanging whole layers. This provides good abstraction and low maintenance. Another benefit is that constructing the software layer-like, breaks complex problems into smaller, more manageable pieces. Since the software is segmented, developing it in teams becomes easier.
When it comes to the downsides, they vary greatly depending on the size of the software solution and complexity. When developing an "over-sized" software package one might loose the perfect overview since none of the actual functionality is revealed. A layered architecture can be a form of *information hiding*. The actual layer should not be aware of the implementation details of another layer in its own operations. This can be done if it leads to better performance.

To find out more about the layered architecture pattern, please read *Pattern Oriented Software Architecture* by F.Buschmann, R.Meunier, H.Rohnert, P.Sommerlad and M.Stal [7].

Part IV

Redesign of Architecture

CHAPTER 13

Introduction

This part describes the new architecture of the Peer2Me framework. Here, you will find the requirements for the Peer2Me framework and how the requirements have been fulfilled.
This chapter will also describe the packages in the framework, along with its classes. We will argue the challenges we faced, along with the solution.

This part is especially relevant to developers wanting to improve the framework.

CHAPTER 14

Requirements

This chapter presents the functional and non-functional requirements of the Peer2Me framework. In the master thesis by Lund and Norum [31] a set of requirements to the old framework is presented. Since this report presents a new and improved version of the Peer2Me framework, the requirements have been slightly modified. The requirements are based on the previous work by Lund and Norum [31] and the evaluation of the Peer2me framework presented in our depthstudy [5]. See Chapter 20 for a comparison of the two sets of requirements.

14.1 Functional Requirements

This section presents the functional requirements of the Peer2Me framework. Functional requirements are a set of instructions reflecting the functionality which must be implemented in the application. The requirements are presented in Table 14.1.

Additional information about the functional requirements

FR 14 - At first glance, this requirement might seem trivial, but it is highly necessary. During the application development phase in our depthstudy [5], we discovered that logging is harder on mobile phones than on desktop computers. When running MIDlets in an emulator on the computer, a console is always available and the Java method: `System.out.println("Some debugging text")` can be used. However, when deploying the application on a real phone, the console is not available. This inspired us to develop a module that can be used to debug applications running on a real phone. The applications PeerQuiz and PeerShare in our depthstudy [5] use this module, and we plan to implement this module in the new Peer2Me framework.

Explanation of terms used in the functional requirements

Node - A node is a mobile phone that runs the framework within an application. This node can be a part of a larger ad hoc network.

Group - A collection of nodes that are aware of each other. The nodes in the group can communicate with each other.

FR 1	The framework must support mobile phones
FR 2	The framework must be able to connect to other nodes
FR 3	The framework must support creation of ad hoc networks
FR 4	The framework must be able to connect to an existing ad hoc network
FR 5	Nodes in the network must be able to exchange messages
FR 6	It must be possible to store primitive variables (String, int, double, long, etc) in a message
FR 7	It should be possible to store serialized java objects in a message
FR 8	It must be possible to attach files to a message
FR 9	It should be possible to store primitive variables, serialized objects and files in one message
FR 10	The framework must track the origin of a message in order to enable a messagereply
FR 11	The framework must be able to create an open group
FR 12	The framework must simulate a pure P2P network
FR 13	The framework must hide the underlying network technology
FR 14	The framework must offer an easy way to create a log that can be viewed in the mobile phone during runtime
FR 15	The framework should register when a nodes leaves the network

Table 14.1: *Functional requirements*

Bluetooth networking - A Bluetooth network must have one master and up to seven slaves. This does not represent a pure peer-to-peer network where all nodes have equal roles and responsibilities. It would be a great feature of the Peer2Me framework to hide the underlaying network technology from the application.

14.2 Quality requirements

An important thing to keep in mind when designing applications, is to consider the non-functional aspects; The quality requirements. They are often just as important as the functional requirements. This section describes the quality requirements we would like the Peer2Me framework to have. The following roles is mentioned in this section and it is important for the reader to distinguish between these:

- **The framework developer -** The developer that is responsible for changes and modifications to the framework.
- **The application developer -** The developer that use the Peer2Me framework in his/hers application.
- **The user -** The person that use the application developed by the application developer.

14.2.1 Modifiability

The Peer2Me framework should be designed in a way that allows future modifications and/or additional modules.

M1 – Implement a workable network component in Peer2Me	
Source of stimulus	The framework developer
Stimulus	The framework developer wants to implement a workable network component in the framework other than Bluetooth
Environment	Design time
Artefact	The network layer in the Peer2Me framework
Response	The framework developer should only have to implement a network component on the same architectural level as the Bluetooth technology without modifications to other parts of the framework
Response Measure	Persuming that the network component works, it should not take more than a couple of days to implement.

14.2.2 Usability

This section concerns usability. Normally usability is associated with the end-user of a system. However, in this context, usability is concerned with how easy and convenient the framework is to utilize for the application developer.

U1 – Inform the application developer of exceptions	
Source of stimulus	The application developer
Stimulus	An internal exception in the Peer2Me framework is thrown
Environment	Runtime
Artefact	The application
Response	Calls to methods that might throw an exception must be placed in a ”try-catch” block in the application
Response Measure	The correct exception should always be thrown and the framework should not block or become unavailable in 95 % of the time

Proper exception handling is crucial for the usability of the framework.

U2 – Aid the implementation of Peer2Me i applications	
Source of stimulus	The application developer
Stimulus	The application developer is implementing exposed framework-methods into his/her application
Environment	Design time
Artefact	The developers application
Response	Peer2Me Code documentation should support well known Javadoc-standards in order to aid the developer when using the framework (code insight etc)
Response Measure	All classes, interfaces, methods and public variables must be well documented with Javadoc

Descriptive naming of packages, classes, interfaces, methods and variables helps the application developer to easily utilize the framework.

14.2.3 Testability

This section concerns how the Peer2Me framework can be tested during development. During the development phase in our depthstudy [5], we discovered that if a MIDlet runs perfectly on the emulator, it might not run at all on a real phone. Most emulators do not act in the same way as a real phone, especially when running MIDlets that use the JAWBT API. By experience, we have discovered that network connections on real phones are harder to establish and harder to maintain. Since networking is a essential part of the Peer2Me framework, it must be possible to debug the framework on real phones and not just the emulator.

T1 – Log	
Source of stimulus	The framework developer
Stimulus	The framework developer wants to debug the framework on real phones using a log
Environment	Runtime
Artefact	The Peer2Me framework
Response	The log should be available on the mobile phone
Response Measure	The log should always be accessible on the mobile phone

Since a mobile phone does not have a console displaying debug information, a log functionality recording events in framework would prove useful.

14.2.4 Availability

This section describes some important availability scenarios. Since the framework establishes network connections, availability is crucial for the success of the framework.

A1 – Normal disconnection of node	
Source of stimulus	A node disconnects from the network
Stimulus	A node disconnects from the network and informs the other nodes of the disconnection before it happens.
Environment	Runtime
Artefact	Network module
Response	The other nodes in the network registers that the node disconnects
Response Measure	The other nodes registers the disconnection immediately

Scenario A1 should be considered as a very normal scenario that happens often. In a peer-to-peer network, nodes will arrive and leave sporadically.

A2 – Abnormal disconnection of node	
Source of stimulus	A faulty node disconnects from the network
Stimulus	A node fails and disconnects from the network
Environment	Runtime
Artefact	Network module
Response	The other nodes in the network registers that the node disconnects
Response Measure	The other nodes registers the disconnection within 30 seconds

Scenario A2 occurs when something goes wrong with a node which causes it to disconnect from the network. This can be anything from the battery depleting or other events that are out of the control from the application environment. The time it takes for the other nodes to find out that a node is missing, cannot be set too strict, since it would require a lot of resources. Therefore, we have required that the other nodes should discover that a node is missing within 30 seconds.

A3 – A new node arrives in an established network	
Source of stimulus	A new node
Stimulus	A new node wants to join the network
Environment	Runtime
Artefact	Network module
Response	The new node searches the network and informs everyone of its arrival
Response Measure	The time it takes for a new node to search and send messages is highly dependant on the mobile phone's performance and network technology. Testing have shown that a Sony Ericsson K750i uses approximately 10 seconds to perform this task using Bluetooth.

CHAPTER 15

Architecture Design Decisions

Before the actual implementation of a software solution is started, many important pieces must be laid out in order to get oversight and structure. As well as requirements, a base design is important to get a rigid and easy-to-maintain construction. When the base design is roughed out and implemented, one must keep in mind to apply well-known, good working tactics and patterns when applicable. Getting all this right, will result in software which is easy to understand and maintain.

This chapter discusses the many decisions made about how the framework could be designed. It elaborates why the different pattern and tactics are used, and in some cases, possible alternative methods. This is presented in a collected manner to make it easier to understand the crossroads we have encountered, and how we dealt with these. This is done over next sections.

15.1 A Pure P2P Model

As mentioned in Section 8.1, there are three basic peer-to-peer models, each laying out the structure a bit differently. Choosing either a *hybrid* or a *mixed* model will involve great limitations to the framework. Both the hybrid and mixed model needs a central unit which keeps information about the nodes resulting in a classic client/server, or more appropriate, a master/slave configuration. To keep awareness and maintain availability in such a configuration is very difficult; If the master-node looses its connections, it will result in a dead network. A solution would be to delegate the server responsibility to a new node, but the algorithms required in such a task would be complex.

We chose to implement the framework based on a *pure P2P model* disregarding any client/server or master/slave solutions. In a pure P2P network, each node is equal and shares the same responsibility. Every node knows every node directly, not through a master node which would be the case in a hybrid network. This means that if one node looses its connections, the network is still alive without the need of delegating new responsibility. Detecting node disconnections and node loss is also much simpler and less resources demanding. This is covered in Section 15.5, and provides a robust and stable network.

Another great advantage with a pure P2P network is the possibility of designing *scatternets*, described in Section 8.5. This means that two nodes do not need to be in direct connection with each other to communicate. The network can therefore cover larger areas and include more nodes.

15.2 Layered Software Structure

When designing and implementing a software solution like a framework, it is crucial to have a good architectural structure before implementing all the functionality. Drawing guidelines, defining responsibility and boundaries are important if the result is to be easy-to-follow and maintainable.
Creating large systems like a framework has been done before, and it is smart to search for a *best practice*. One can compare a *pattern* to a best practice. A well known method for solving a known problem. Using an existing pattern will result in a shorter development time and a more rigid solution.

For the main architecture, we have used a *layered architecture pattern*, as described in Section 12.8.1. This gave us a good start designing the structure. Since one of the key elements to the framework is the possibility to exchange parts, particularly the network medium, a layered structure is well suited. This version of Peer2Me is implemented using *Bluetooth*, and it is not required to alter any code to use a implementation of another network technology. Figure 15.1 shows the layered layout of the Peer2Me framework.

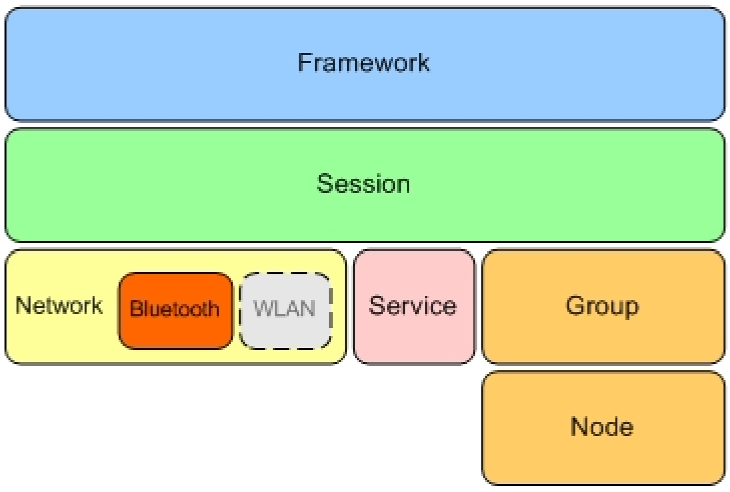


Figure 15.1: *The layered layout of Peer2Me*

As you can see, and as explained in Section 12.8.1, each layer has its own responsibility and only knows the implementation details of the layer over and the layer beneath, there is no horizontal link.
As an example, if *Network* needs something from *Group*, it could not go directly to Group because it has no instance of it, nor has Group any static methods exposed. The only classes that have any knowledge about Group is *Session* (and *Node*). Network knows about Session, and Session knows about Group. *The layered structure defines the information path.*

There are several other structural patterns that might have been suitable, such as the *process structure* or the *client-server* pattern, but do not meet the requirements of Peer2Me. These are described in *Software Architecture in Practice* [3]. The process structure has a good component based structure, which could have made the framework smaller with optional features, but the functionality would be hard to implement. If the client-server should have been applied, the network medium would have been locked and the possibility of a pure P2P model eliminated.
The layered structure have proven to be quite successful, *preventing ripple effects*, leaving the framework easy to maintain. The tactic *prevent ripple effects* is explained in Section 12.3.1.

As Figure 11.1 shows, Java is also a layered structure. How the Peer2Me framework is connected to the Java structure is shown in Figure 16.7.

15.3 Communication protocol

Peer2Me is currently implemented with one network technology, Bluetooth. Bluetooth was chosen because an infrared peer-to-peer network is completely unpractical and there are no WLAN API, available for J2ME. There are currently two protocols that can be used to implement Bluetooth in J2ME: OBEX and RFCOMM. See Section 11.3.1 for further information about these two protocols. Both protocols have their advantages and disadvantages. The previous version of Peer2Me, developed by Lund and Norum [31], used the RFCOMM protocol. We chose OBEX. OBEX is a well developed protocol with much more flexibility than the basic RFCOMM protocol. It allowed us to attach headers that described the data being transported between the nodes, rather than just the data itself. The previous version of Peer2Me appended descriptions all around data being transported. This required special parsing of the data on the receiving node. This has proved to be a very awkward way of transporting data. In our depthstudy [5] we encountered serious problems with this method. It was not possible to send data that contained special characters such as \n (linebreak), and adding several elements to a message did just not work good enough with the RFCOMM protocol. The use of OBEX has allowed us to create and send very advanced messages, containing serialized objects, primitive datatypes and files. We are proud to say that adding more description to the messages and creating new type of messages should be easily accomplished for other developers who want to continue our work.

15.4 Messages

Before we chose the communication protocol, we discussed what kind of functionality we would like the messages to have. We came up with much higher demands than the previous version of Peer2Me had. We also sketched out how the code should look like for an application developer who wants to send a message containing different elements. The previous version of Peer2Me did not provide a user friendly way of sending messages. The application developer had to create a Message object, then create MessagePart objects and then add these to the Message. Our solution does not require the developer to create and add such parts to the message, thus increasing the usability of the framework.

In the previous version of Peer2Me, the developer had to add a message part to a message that contained an `int` in order to identify the type of message on the receiving node. The receiving node then had to retrieve the message part and the retrieve a field value in that messagepart to identify the entire message. This was also awkward. We had to find a way of distinguishing messages, both the messages that the framework creates and sends internally and the messages that the application developer creates and sends. We decided early on that messages sent by the framework internally such as "node joined", "node left" and "node moved" should be separated by messages sent by the application developer. These are the alternatives we were faced with:

1. Use OBEX headers to identify internal messages and add a mechanism that allows the application developer to identify a message using an int.
2. Use OBEX headers to identify internal messages and let the developer decide how he wants to identify messages.

Alternative 1 would force the application developer to use a specific primitive variable to identify messages. Alternative 2 does not lock the application developer to a certain way of defining messages and offers much more flexible solutions than the previous version of Peer2Me and the other alternative.

In our new version of Peer2Me, the developer now has the freedom to distinguish messages in a variety of ways. He can use any of the following as an identifier on a message: `int`, `double`, `float`, `String`, `boolean`, `char`, `long` and `short`. And there is no restrictions if he wants some kind of "sub identifying" scheme either. Listing 15.1 shows how to add a custom identifier to a message. This example uses an int.

Listing 15.1: *Sending two messages identified by an int value*

```
//Creating and sending two messages
Message message = new Message();
message.addElement(1,"type");
message.addElement("This is type 1 message","info");
message.addRecipient(node);
framework.sendMessage(message);

Message message2 = new Message();
message2.addElement(2,"type");
message2.addElement(true,"info");
message2.addRecipient(node);
framework.sendMessage(message2);
```

Listing 15.2 shows how to identify the two different messages sent in Listing 15.1.

Listing 15.2: *Receiving a message*

```
//Receiving a message
public void messageReceived(Message message)
{
 int type = message.getInt("type");
 switch (type)
 {
 case 1:
  String info = message.getString("info");
  break;
 case 2:
  boolean info = message.getBoolean("info");
  break;
 default:
  break;
 }
}
```

If the developer wants to, he can also use a `String` as an identifier of the message or maybe a just a `boolean` if there only exists two types of messages in his application.

The last crossroad we were faced with regarding the message system, was how to prioritize messages. Early testing showed that it was not possible to send more than one message at a time on a specific Bluetooth channel [1]. This meant that if the application tried to send

[1]Bluetooth assigns a connection to a specific channel when it is created. Bluetooth is also restricted to seven simultaneous channels

a message before the previous one was finished, the Bluetooth API would complain that the Bluetooth module was busy at the time. To solve this problem we implemented a queue of dynamic size and a module that checks the queue at regular intervals and processes the queue when messages are added to the queue. This means that the MIDlet that uses Peer2Me does not invoke the sending of the message directly, but just adds the message to a queue which the framework processes. Section 12.3.3 describes scheduling tactics, where FIFO is one of them. We decided to use the FIFO algorithm for handling messages. This eliminates the possibility of low prioritized messages being unprocessed. Changing the queue handling algorithm or modifying it can easily be accomplished if a developer wishes another type of tactic.

15.5 Detecting Node loss

A big challenge when designing mobile applications is availability. It is crucial to have good awareness between the mobile phones, and at all time keep track of the nodes that are available. To achieve this, one can implement different tactics. The two most commonly used tactics are the *ping/echo*-tactic and the *heartbeat*-tactic.

We chose to implement the ping/echo tactic, as this is more reliable. While in the heartbeat-tactic the nodes sends out a message at a given interval, it requires the other nodes to constantly listen for special messages containing the heartbeat, without generating a conformity message to the heartbeat. In a normal situation this tactic would require the least number of messages, hence reducing the network traffic.
The number of messages in a normal situation, ping/echo vs. heartbeat:
(n is the total number of nodes)

- **Ping/echo** 2 x (n-1)
- **Heartbeat** *(n-1)*

As mentioned, this would be in a normal situation. Since we implemented the framework using the *OBEX* protocol instead of the *RFCOMM* protocol, as mentioned in Section 15.3, there is a much more clever way to achieve this. Since it is impossible to send a message to a node that is *not* connected, sending the actual ping-message is therefore unnecessary. Instead of sending the actual message, we can just try to connect and if that is successful the node is alive. If the connection failed, we know that node is currently unavailable. Since there might be other issues preventing the node from accepting a connection, the connection will be retried established a given number of times. As mentioned in Section 16.9, each node carries statistics regarding failed connections and successful ones. A simple call to the method `boolean isNodeAbscent()` will return whether the limit of failed connection is reached or not. Since OBEX includes a standard *headerset* with each connection, the amount of data needed in this communication is much lower than sending a message with i.e. RFCOMM. Using this approach results in *0* ordinary messages, only the headers, and outruns the heartbeat tactic by a large margin.

As the ping/echo tactic runs constantly, it uses its own channel in OBEX, not interrupting the normal communication.
When a node is detected lost, it is removed from the list of available nodes at the node which detected the loss. The loss of a node is *not* broadcasted.

CHAPTER 16

Design Overview

This chapter presents a logical overview along with process views where appropriate of the Peer2Me framework.

Our depthstudy [5] revealed weakness and lack of functionality in the previous version of Peer2Me, resulting in this redesign. First we give a quick overview of the central concepts of Peer2Me, how Peer2Me relate to Java, and then present an architectural overview and a detailed description of the packages and the classes.

16.1 Domain concepts

To fully understand how the framework is designed, a few key concepts must be explained. This section will described these concepts.

Framework: The top entity of the framework and the interface between the application using the framework and the framework itself.

Session: May be regarded as the main link in the framework. A session keeps track of known peers, groups, available network mediums etc.

Service: Identifies the application running the framework.

Network: An abstraction of the network layer representing the communication medium.

Node: A representation of a peer running the framework.

Group: Represent a collection of *node*s known to each other.

Message: An entity that can be exchanged between the nodes. Can be sent to single nodes or to groups. Can contain text, serialized objects or binary data such as files, pictures etc.

Application: The MIDlet using the framework.

None of the concepts above are directly dependent of a specific network medium, but might have small limitations depending on the network technology. It should not be necessary to alter the conceptual design when changing network mediums.

Figure 16.1 illustrates how the domain concepts are related to each other.

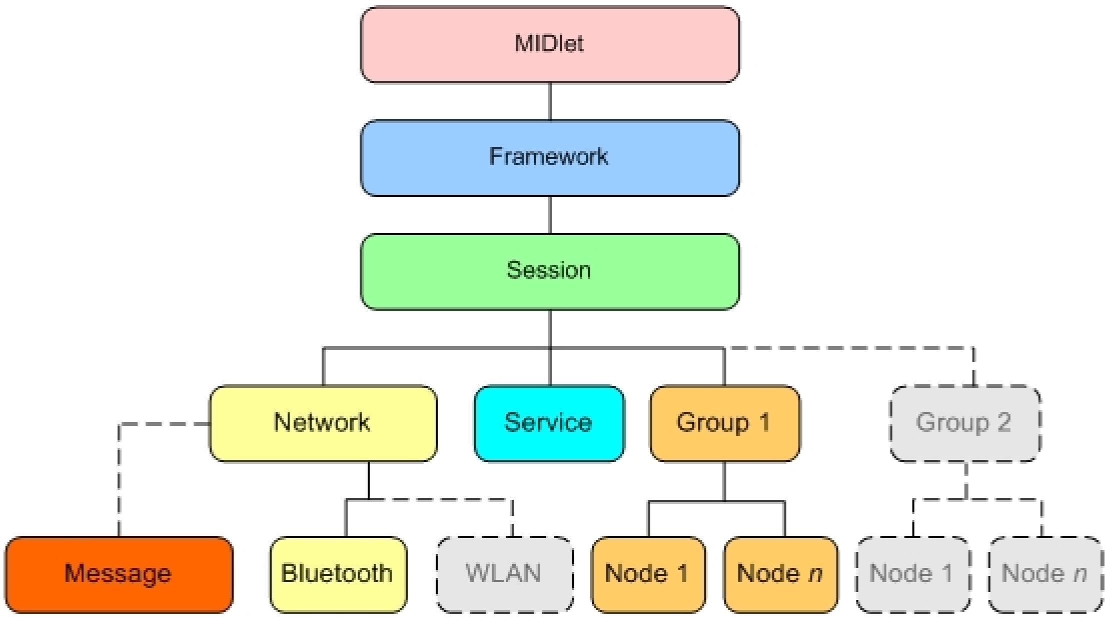


Figure 16.1: *Domain concepts*

16.2 High level general process scenarios

This section presents how data is passed between the different layers in the Peer2Me framework in some typical scenarios. This section also presents how nodes change roles depending on which node initiates a transfer of a message.

The MIDlet interacts with the framework through the *Framework* class. When the Peer2Me framework wants to send data to the MIDlet, it does this through the *FrameworkSubscriber*. Figure 16.2 shows how the Peer2Me is initialized when the MIDlet calls the method `initialize()` on the *Framework* class.

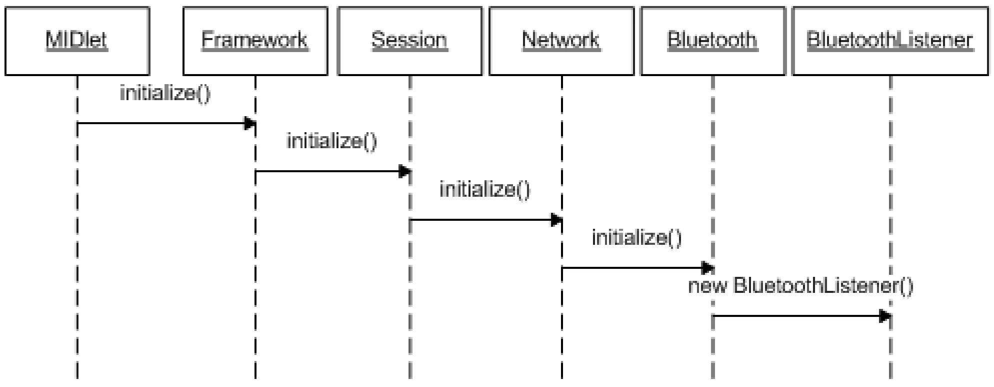


Figure 16.2: *Initialization of the Peer2Me framework*

After the Peer2Me framework has been initialized, the MIDlet can start a search for other nodes. Figure 16.3 shows how the framework initiates a search through the different layers and how the framework alerts the MIDlet when a node is found.

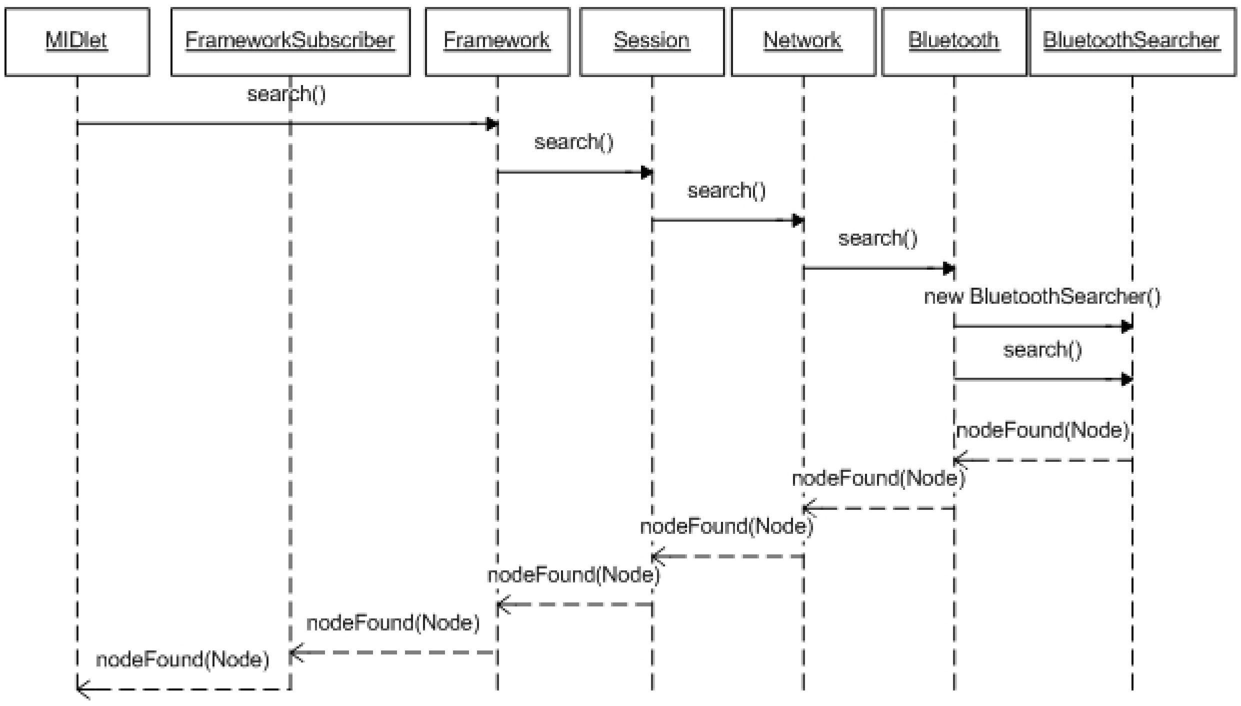


Figure 16.3: *Search for other nodes*

After a search has completed, a node can send a message to another node. Since we have implemented a pure peer-to-peer Bluetooth network, nodes are independant of a central node and must be able to change ”roles” depending on the situation. Figure 16.4 shows three nodes which are all aware about each other, but none of them have been assigned either a master or a slave role.

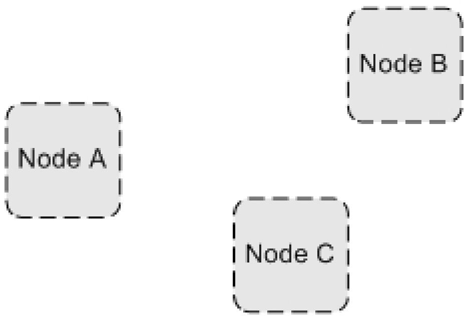


Figure 16.4: *None of the nodes has been assigned a role*

If node A wants to send a message to node B, node A will take the role as master and node B becomes a slave, see Figure 16.5.

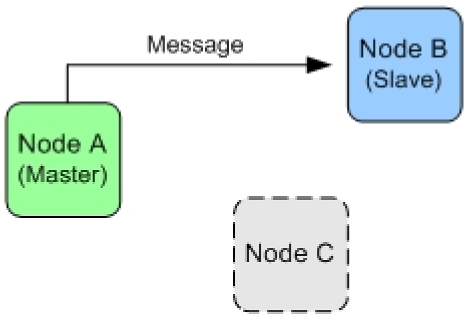


Figure 16.5: *Node A sends a message to node B*

Later, if node B wants to send a message to node C, node B will become the master of that connection. The transfer of a message from B to C is illustrated in Figure 16.6.

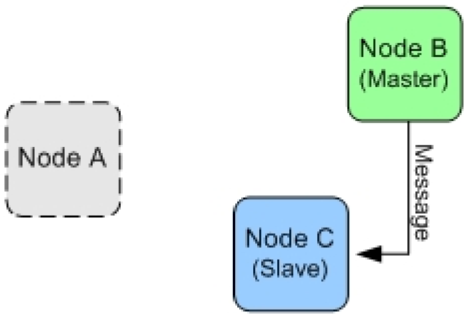


Figure 16.6: *Node B sends a message to node C*

16.3 Peer2Me and Java

This section describes how the Peer2Me framework relates to Java. As the previous version of Peer2Me, this version also use the Java 2 Mobile Edition SDK as described in Section 11.2 and is designed to run on mobile phones supporting MIDP 2.0. As Figure 16.7 shows, the framework lies between the base J2ME components and the MIDlet application.

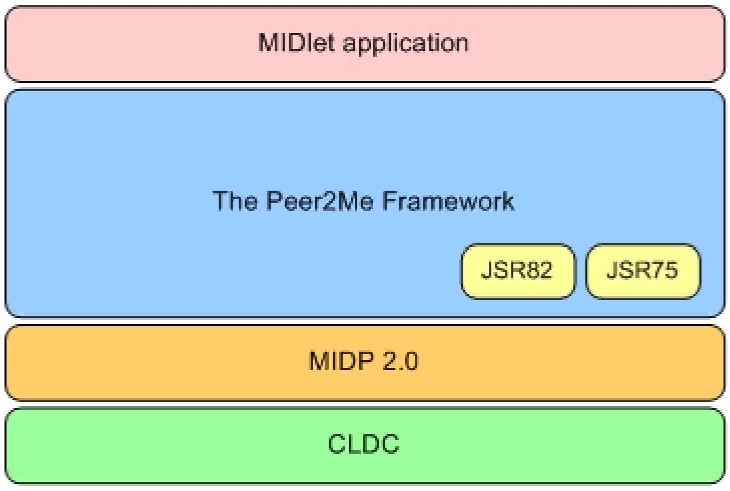


Figure 16.7: *How the framework relates to J2ME*

The MIDlet profile MIDP 2.0 defines the methods and objects available to the framework implementation, running on top of the CLDC, as described in Section 11.2. The application using the framework, the MIDlet, lies on the top.

As shown in the figure above, two optional packages provided by Java is required in the framework. The JSR75 which is used for accessing PIM data and the filesystem, and the JSR82 which is the Java API for Bluetooth.

16.4 Peer2Me Package Overview

This section will give an overview of the packages in Peer2Me, while the preceding sections will give detailed information about each package. Figure 16.8 shows an overview of the packages found in Peer2Me, and how they are hierarchically ordered.

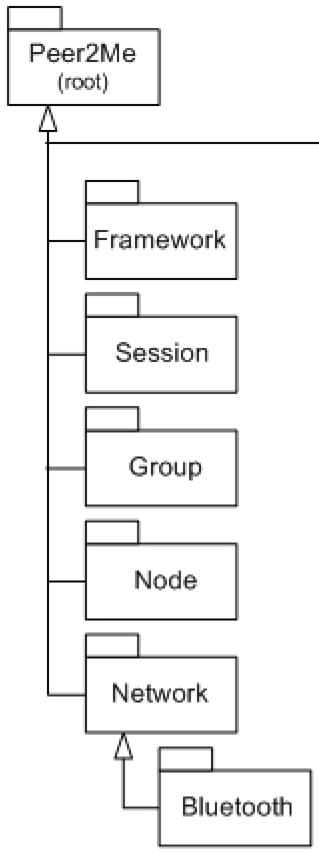


Figure 16.8: *The packages of Peer2Me*

Each of the packages displayed in Figure 16.8, relates to its own layer-component in Figure 15.1 found in Section 15.2, except *Log*, *Message*, *Util* and *Exception*. As you can see, the *Bluetooth* package is subservient the *Network* package.
We will now present each package.

16.5 The Framework package

The framework package is the main link between the Peer2Me framework and the application. It resides above all the other packages in the layered structure.
One key element in this version of Peer2Me is that the developer using the framework should deal with fewest possible classes from the framework, keeping the exposed methods tidy and understandable. The developer will normally only use methods provided by the framework, message and node classes. Even though many of the classes in the framework are `public`, they are not intended to be used outside the framework.

This package provides one class, *Framework* and one interface, *FrameworkSubscriber*. Both equally important and used by each other. They are explained in the preceding sections beginning with Section 16.5.1.

16.5.1 Class: Framework

Applications using the Peer2Me framework will interact mostly with this class. After this class is instantiated, it creates an instance of *Session*, initializes the *Log*, sets the *FrameworkSubscriber* and the *FileHandler*. The framework class uses the principle of the *singleton design pattern*, and is created as shown in Listing 16.1. Singleton is described in Section 12.6.1.

Listing 16.1: *The Framework is created in an application*

```java
//Creates a new instance of Framework
public static synchronized Framework getInstance(String groupName,
  String serviceName, Network network, boolean activatePing,
  FrameworkSubscriber frameworkSubscriber)
 {
  if(singleton == null)
  {
   singleton = new Framework();
   singleton.setSession(Session.getInstance(singleton.getFramework(),
     new Group(groupName), new Service(serviceName),
     network, activatePing));
   singleton.setframeworkSubscriber(frameworkSubscriber);
   singleton.setLog(Log.getInstance());
   singleton.setFileHandler(FileHandler.getInstance());
  }
  return singleton;
 }
```

As you can see in the listing above, `Framework.getInstance(..)` is needed to get an instance of *Framework*, to which `initialize()` must be called. The framework then calls on the `initialize()` method in *Session*, which again use `initialize()` in *Network*. This approach, a *strict information path* in a *layered architecture*, is covered in Section 15.2. This initializes the framework, which then starts listening for incoming connections. The framework awaits further instructions from the application, which can be to initiate a search for other nodes. The framework can now also accept incoming connections. It must also be decided when calling on the `Framework.getInstance(..)` if the *ping functionality* should be activated or not. This is controlled by `boolean activatePing`. When the framework starts a search with the method `search()` it forwards the call to *Session*, *Network*. framework will now use the network technology provided in the `getInstance(..)` method call, to search for other devices running the framework with the appropriate *ServiceID*. The principle behind a serviceID is described in Section 11.3.1 and Section 16.7. When a search is finished, the method `searchCompleted()` is invoked.

Besides the two main methods, `initialize()` and `search()`, the framework class offers methods for sending messages, handling groups, nodes etc. These methods are all forwarded through *Session* to the rest of the Peer2Me framework, but gathering them in the framework class using principles of the *facade* pattern, provides a better overview for the developer using the framework. The facade pattern is described in Section 12.7.1. When the application sends a message, it must pass on an object of the type *Message* to the method `sendMessage(Message message)`. How a message is constructed is explained in Section 16.12.1. The message will then be passed through the layers.
The framework will at all time store discovered nodes, their groups and all associated information. It is therefore unnecessary to store connected nodes in the application, and instead use the provided *get* methods for retrieving nodes and groups. A detailed description of these

methods can be found in their respective packages. Methods for handling a *Group* are found in *Session* and methods for handling a *Node* are found in *Group*.

There are five methods in the framework class which depends on an interface to be implemented by the application. The interface *FrameworkSubscriber* must be implemented by the application using the framework, and give a reference to the framework via the method `setFrameworkSubscriber`. Not doing this before engaging any communication will result in an exception. The interface is described in Section 16.5.2.

16.5.2 Interface: FrameworkSubscriber

This interface is the connection between the application and the Peer2Me framework. It **must** be implemented by the application, forcing it to implement five simple methods, showed in Listing 16.2. This mechanism is based on the *observer* pattern described in Section 12.5.1.

Listing 16.2: *The methods in the FrameworkSubscriber needed by Framework*

```java
public void nodeDiscovered(Node node)
{
 //node -> The node that is discovered
}

public void nodeLost(Node node)
{
 //node -> The node that is lost
}

public void messageReceived(Message message)
{
 //message -> The message that is received
}

public void messagePartReceived(String messageID, int part, int total)
{
 //part & total -> part of total received
}

public void searchCompleted()
{
 //Search is completed
}
```

Details on the objects passed to the methods can be found in Section 16.12.1 for *Message* and Section 16.9 for *Node*.

16.5.3 Class Diagram

This section show the class diagram of the framework package in Figure 16.9.

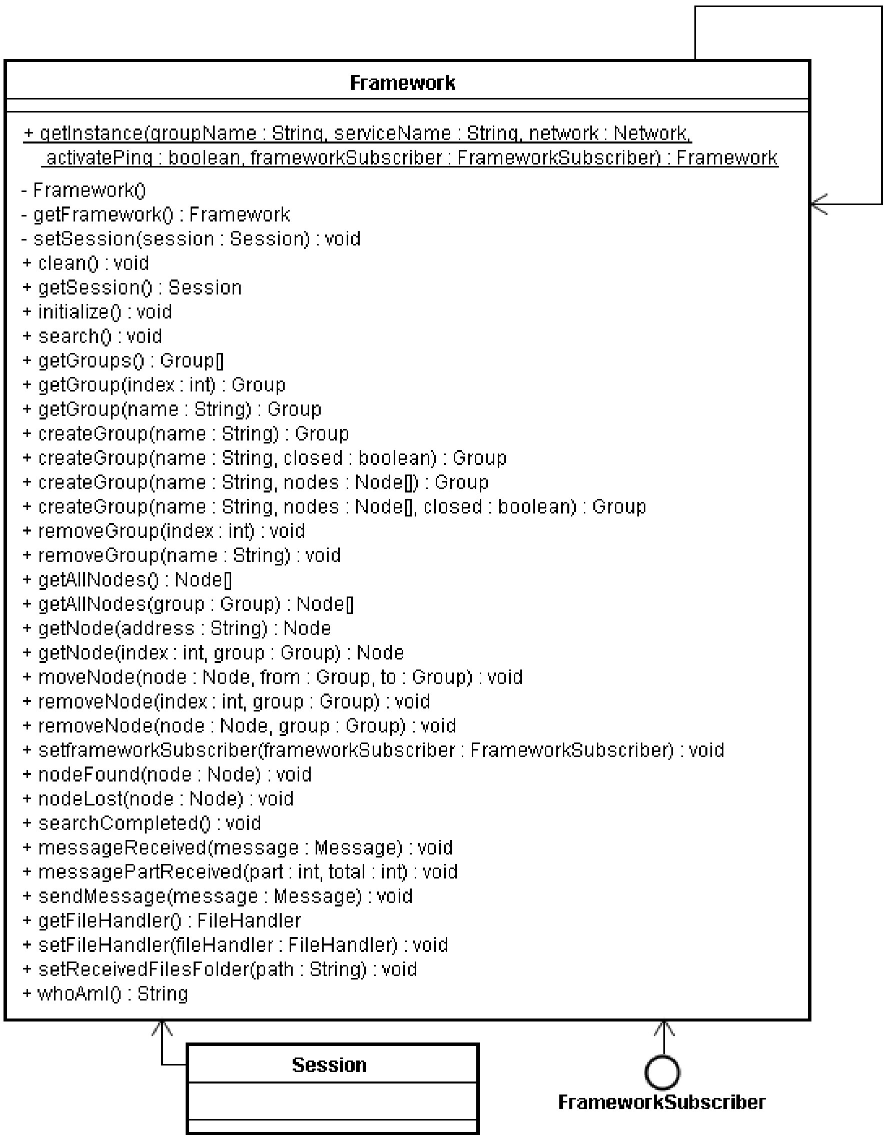

Framework
+ getInstance(groupName : String, serviceName : String, network : Network, activatePing : boolean, frameworkSubscriber : FrameworkSubscriber) : Framework
- Framework()
- getFramework() : Framework
- setSession(session : Session) : void
+ clean() : void
+ getSession() : Session
+ initialize() : void
+ search() : void
+ getGroups() : Group[]
+ getGroup(index : int) : Group
+ getGroup(name : String) : Group
+ createGroup(name : String) : Group
+ createGroup(name : String, closed : boolean) : Group
+ createGroup(name : String, nodes : Node[]) : Group
+ createGroup(name : String, nodes : Node[], closed : boolean) : Group
+ removeGroup(index : int) : void
+ removeGroup(name : String) : void
+ getAllNodes() : Node[]
+ getAllNodes(group : Group) : Node[]
+ getNode(address : String) : Node
+ getNode(index : int, group : Group) : Node
+ moveNode(node : Node, from : Group, to : Group) : void
+ removeNode(index : int, group : Group) : void
+ removeNode(node : Node, group : Group) : void
+ setframeworkSubscriber(frameworkSubscriber : FrameworkSubscriber) : void
+ nodeFound(node : Node) : void
+ nodeLost(node : Node) : void
+ searchCompleted() : void
+ messageReceived(message : Message) : void
+ messagePartReceived(part : int, total : int) : void
+ sendMessage(message : Message) : void
+ getFileHandler() : FileHandler
+ setFileHandler(fileHandler : FileHandler) : void
+ setReceivedFilesFolder(path : String) : void
+ whoAmI() : String
Session
FrameworkSubscriber

Figure 16.9: *Class Diagram of the Framework Package*

16.6 The Session package

I our layer-structured framework, *Session* is the most essential part, joining together all the other classes and controlling the information-flow both upwards and downwards in the hierarchy. As shown in Figure 16.1, the main branches, Group, Network and Service, all arise from *Session*. Doing this, we provide strict rules for how methods can be called upon, variables transported and objects instantiated.

An application using the framework will only have one instance of *Session*. The Session package consists of only one class uniting access methods for operations regarding *group* and *network*.

16.6.1 Class: Session

Session is one of the bigger classes in the framework providing many getters[1] and setters[2], but reducing it into several smaller classes would take away the clarity and uniformity. As mentioned, the framework only need one instance of Session, and Session therefore implements the *singleton pattern*. The singleton pattern ensures that only one instance of the class is created, and the pattern is described in Section 12.6.1. The session class does not contain any complicated logics, but creates a default *Group* and sets the *network* along with some local properties.

[1]Getters - Methods that retrives a specific variable are often referred to as a "getter"-method
[2]Setters - Methods that sets the value of a specific variable are often referred to as a "setter"-method

16.6.2 Class Diagram

This section show the class diagram of the session package in Figure 16.10.

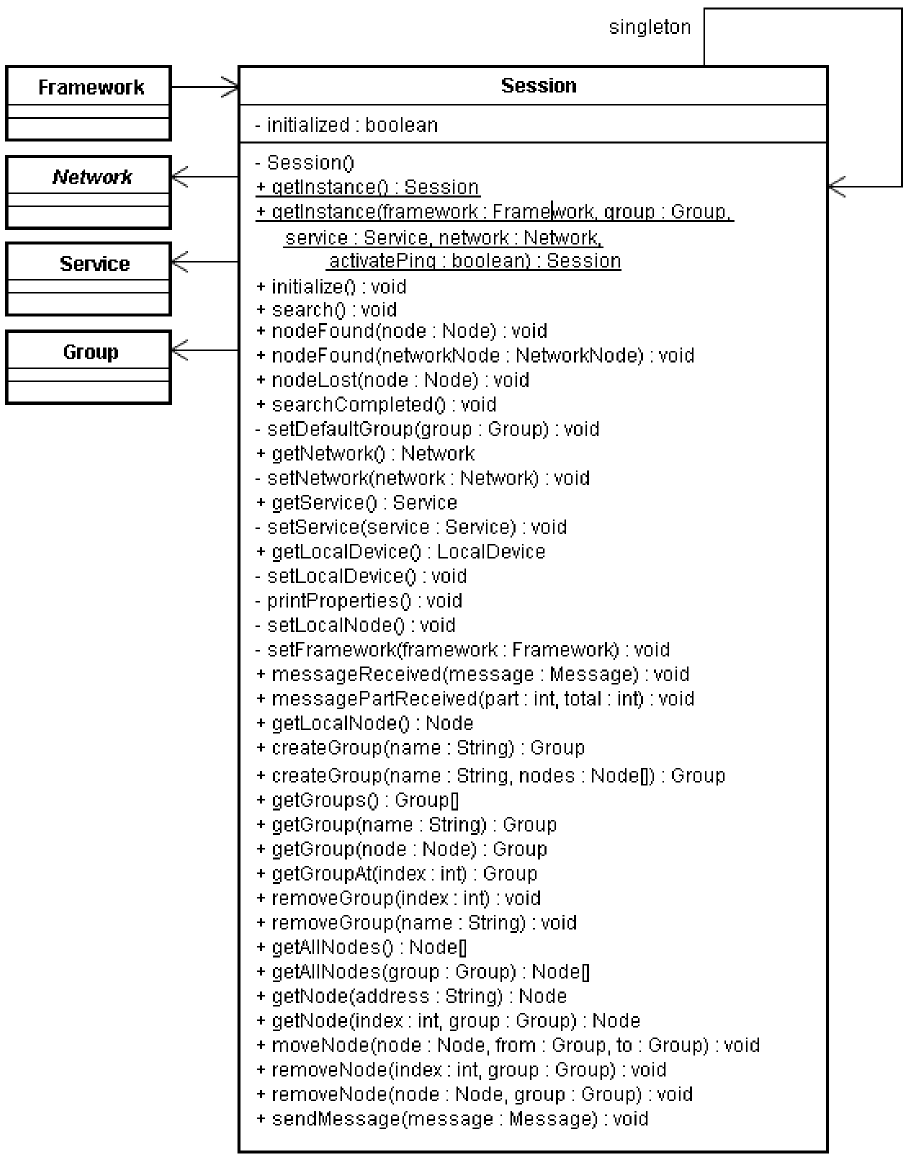

singleton
Framework
Network
Service
Group
Session
- initialized : boolean
- Session()
+ getInstance() : Session
+ getInstance(framework : Framework, group : Group, service : Service, network : Network, activatePing : boolean) : Session
+ initialize() : void
+ search() : void
+ nodeFound(node : Node) : void
+ nodeFound(networkNode : NetworkNode) : void
+ nodeLost(node : Node) : void
+ searchCompleted() : void
- setDefaultGroup(group : Group) : void
+ getNetwork() : Network
- setNetwork(network : Network) : void
+ getService() : Service
- setService(service : Service) : void
+ getLocalDevice() : LocalDevice
- setLocalDevice() : void
- printProperties() : void
- setLocalNode() : void
- setFramework(framework : Framework) : void
+ messageReceived(message : Message) : void
+ messagePartReceived(part : int, total : int) : void
+ getLocalNode() : Node
+ createGroup(name : String) : Group
+ createGroup(name : String, nodes : Node[]) : Group
+ getGroups() : Group[]
+ getGroup(name : String) : Group
+ getGroup(node : Node) : Group
+ getGroupAt(index : int) : Group
+ removeGroup(index : int) : void
+ removeGroup(name : String) : void
+ getAllNodes() : Node[]
+ getAllNodes(group : Group) : Node[]
+ getNode(address : String) : Node
+ getNode(index : int, group : Group) : Node
+ moveNode(node : Node, from : Group, to : Group) : void
+ removeNode(index : int, group : Group) : void
+ removeNode(node : Node, group : Group) : void
+ sendMessage(message : Message) : void

Figure 16.10: *Class Diagram of the Session Package*

16.7 The Service package

The service package provides simple functionality for registering a *serviceID* used when distinguishing different application running on bluetooth. The package consists of two classes described in the sections following.

16.7.1 Class: Service

This class is used to store a *serviceID* and *pingserviceID* in two formats. One in plain ASCII characters, and one in hexadecimal characters. The ASCII representation is the serviceID submitted by the developer using the framework, and used in the *connection url* as described in Section 16.11.2. This is the same for the serviceID used by the ping functionality.
The serviceID represented by hexadecimal characters is derived from the serviceID in ASCII character, by using the *HexBuilder* class. This representation of the serviceID is used as an *Universally Unique IDentifier.* An UUID is said to be a unique identfier of a software application, and can be generated in *J2ME* by the *java.util.UUID* or in *J2ME* by *javax.bluetooth.UUID.* These will only generate either a 128-bit representation, or a 16-bit representation, and it is not garanteed that it will be the same each time, which is needed in the framework. The framework has therefore its own conversion algorithm, found in Section 16.7.2.

16.7.2 Class: HexBuilder

This class cannot be instanciated, and provides only one method,
`static String getHexString(String serviceID)`. This method will convert a ASCII string to a 32-bit hexadecimal representation. If the ASCII string is shorter than 16 characters, the hexadecimal representation will be shorter than 32-bit. It is therefore looped until exactly 32-bit.
If something goes wrong, a *HexConversionException* is throwed.

16.7.3 Class Diagram

This section show the class diagram of the service package in Figure 16.11.

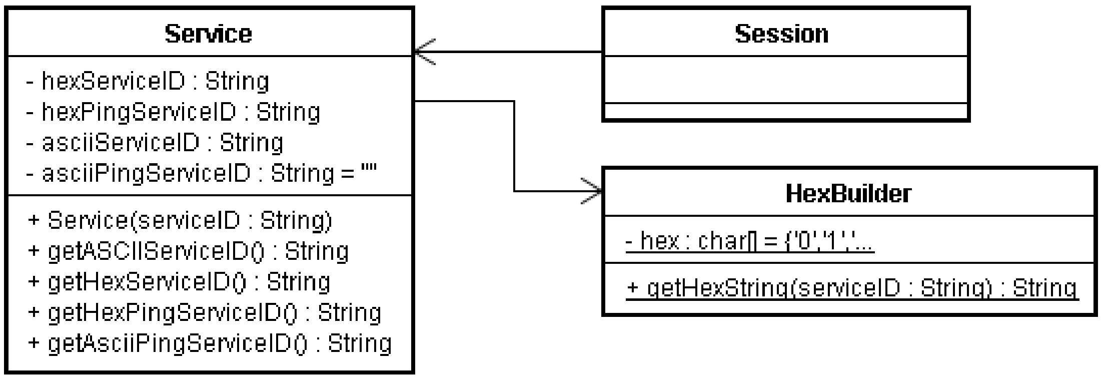

Service
- hexServiceID : String
- hexPingServiceID : String
- asciiServiceID : String
- asciiPingServiceID : String = ""
+ Service(serviceID : String)
+ getASCIIServiceID() : String
+ getHexServiceID() : String
+ getHexPingServiceID() : String
+ getAsciiPingServiceID() : String
Session
HexBuilder
- hex : char[] = {'0','1',...
+ getHexString(serviceID : String) : String

Figure 16.11: *Class Diagram of the Service Package*

16.8 The Group Package

A peer-to-peer network can involve many participants, and can get quite complex. The number of participants can affect performance and increase overhead if not dealt with properly. A way to deal with parts of this problem is to divide the participants into smaller groups to increase the overview and make the network more efficient. One can regard the group-layout as a *best practice*.

The Peer2Me framework is no exception and provides grouphandling. When participants are joining the network, they are put into a default group. They can then afterwards be moved into new groups, join existing groups or be removed from a group. There are no limit to how many participants a group can have.

16.8.1 Class: Group

The group package consists of only one class, *Group.java*. This class stores all the participants, known as a *Node*, in the group. The *Node* class is covered in Section 16.9. The group is identified by a *name*, and contains a vector that holds the nodes. Proper methods are exposed for retrieving, moving and removing nodes. An example of how to move a node from one group to another is given in Listing 16.3.

Listing 16.3: *Example of creating a group and moving a node into it*

```
Group myCurrentGroup = framework.getGroup("MyCurrentGroup");
Group myNewGroup = framework.createGroup("MyNewGroup");
Node myNode = framework.getNode(0,myCurrentGroup);
framework.moveNode(myNode,myCurrentGroup,myNewGroup);
```

As you can see, the method `moveNode(Node node, Group from, Group to)` takes the node you want to move, and the groups used in the transfer as parameters. The method `getNode(int index,Group group)` gets the node, or throws a NodeNotFoundException if the node is not found in the group.

16.8.2 Class Diagram

This section show the class diagram of the group package in Figure 16.12.

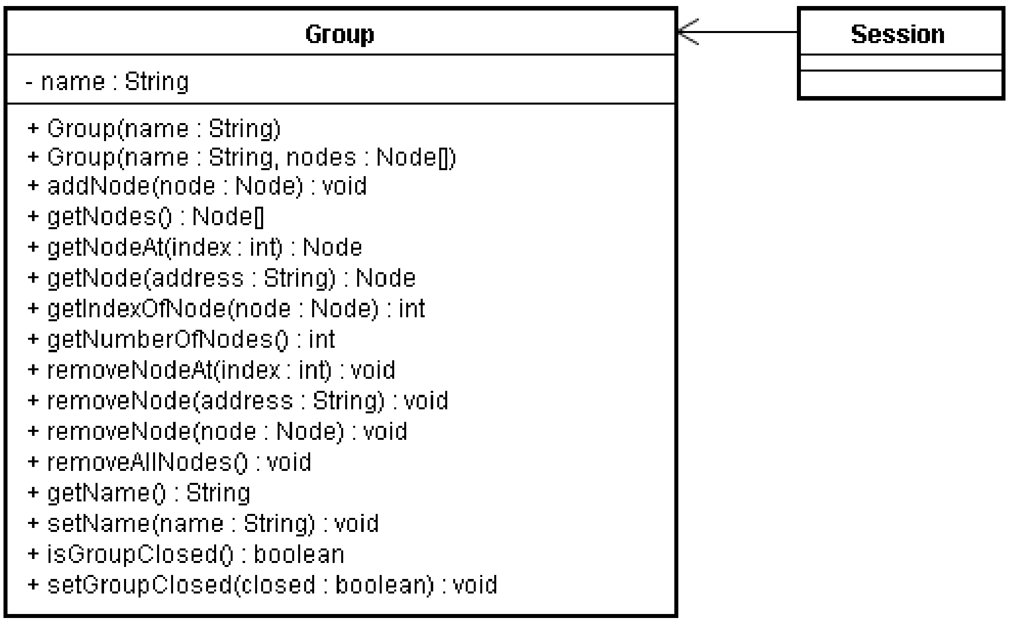

Group
- name : String
+ Group(name : String)
+ Group(name : String, nodes : Node[])
+ addNode(node : Node) : void
+ getNodes() : Node[]
+ getNodeAt(index : int) : Node
+ getNode(address : String) : Node
+ getIndexOfNode(node : Node) : int
+ getNumberOfNodes() : int
+ removeNodeAt(index : int) : void
+ removeNode(address : String) : void
+ removeNode(node : Node) : void
+ removeAllNodes() : void
+ getName() : String
+ setName(name : String) : void
+ isGroupClosed() : boolean
+ setGroupClosed(closed : boolean) : void
Session

Figure 16.12: *Class Diagram of the Group Package*

16.9 The Node package

A peer-to-peer network has two or more participants. The collective term for a participant is *nodes*. A node stores information about the participant, and will in Peer2Me represent a device running an application using the framework. When someone is sending data, he/she is using a *Node* as the representative of the recipient device. The node package contains only one class representing the data structure of a node.

16.9.1 Class: Node

This class stores basic information such as name and address, along with a reference to a *NetworkNode* for a device. The *NetworkNode* class is described in Section 16.10. Simple statistics regarding the ping/echo-functionality is also stored here. The node keeps track of how many requests (ping) it has responded (echo) to.
A node is created based on the *NetworkNode*, which is an abstract class. In this masterthesis, it would typically be a *BluetoothNode*, described in Section 16.11. This means the node is in this case initially based on a *RemoteDevice*, also described in Section 16.11. A node is **only** created by the framework, and should **not** be created by the developer using the framework. The nodes are created in *Session.java*, as shown in Listing 16.4.

Listing 16.4: *Codesnippet from Session.java where node is created*

```
public void nodeFound(NetworkNode networkNode)
{
 Node node = new Node(networkNode);
 framework.nodeFound(node);
 group.addNode(node);
}
```

16.9.2 Class Diagram

This section show the class diagram of the node package in Figure 16.13.

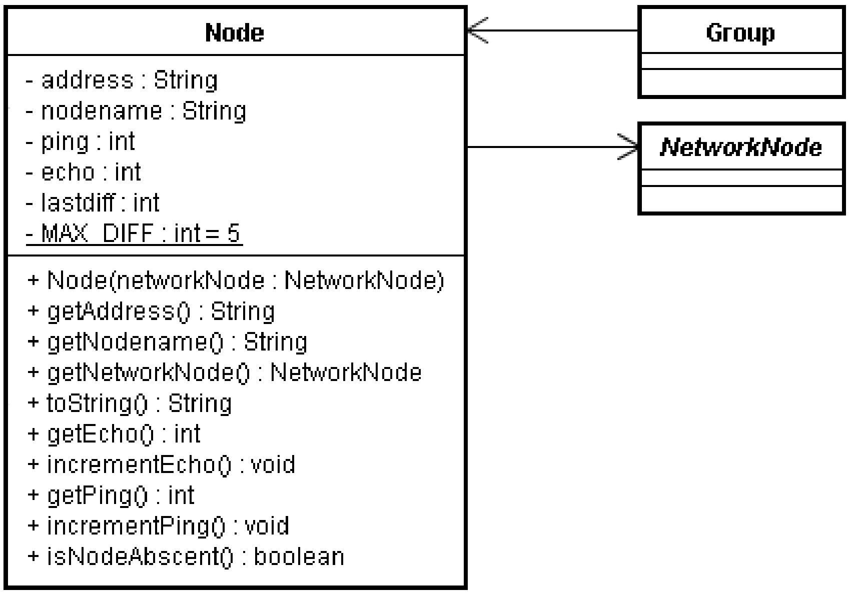


Figure 16.13: *Class Diagram of the Node Package*

16.10 The Network package

A feature in Peer2Me is to abstract the network layer, making the communication semi-transparent for the developers. An important feature is also to make the network layer modular, in other words easy to switch the network technology with another. In this version of Peer2Me, Bluetooth is implemented as the network carrier, but can easily be exchanged with e.g. WLAN instead. This is because the actual framework components are implemented towards abstract classes defining a set of required methods. The base network components could have been implemented as interfaces, but in this case it is not recommended.
A simple hierarchical layout of *Network* is shown in Figure 16.14.

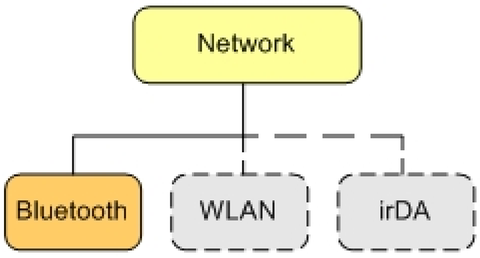


Figure 16.14: *A simple hierarchical layout of network*

The Network package consist of three components; the *Network* class, *NetworkNode* and *NetworkTimer*, as described below.

16.10.1 Abstract Class: Network

This class must be inherited by the desired network technology's base class. This class defines a set of methods such as `search()`, `sendMessage(Message message)`, `nodeFound(Node node)` etc., which are required implemented by all network technologies. In the current version, *Bluetooth* extends this class. In this case, *Network* is inherited by *Bluetooth* as shown in Listing 16.5. *Bluetooth* is covered in Section 16.11.

Listing 16.5: *The Bluetooth-class inherits the abstract Network-class*

```
import peer2me.network.Network;

public class Bluetooth extends Network
{
```

16.10.2 Abstract Class: NetworkNode

This class describes a set properties required in the node representation used by *Node* covered in Section 16.9. NetworkNode must be inherited by the node-class used by the particular network technology. In this case, *NetworkNode* is inherited by *BluetoothNode* as shown in Listing 16.6. *BluetoothNode* is covered in Section 16.11.

Listing 16.6: *The BluetoothNode-class inherits the abstract NetworkNode-class*

```
import peer2me.network.NetworkNode;

public class BluetoothNode extends NetworkNode
{
```

Since NetworkNode only defines a simple set of methods, thereby the method `getObject()`, it gives the developer of a new network technology freedom of adding methods and variable to the class *inheriting* NetworkNode. The options the datatype *object* in NetworkNode provide, enables the developer to embed complex datatypes in to NetworkNode, or in this project BluetoothNode. And as NetworkNode is abstract and has no predefined constructor, the class inheriting NetworkNode can be created in many ways offering flexibility.

16.10.3 Class: NetworkTimer

Extends: java.util.TimerTask

NetworkTimer is used to create a ticker firing off at a given interval. It controls the interval between each ping-message, running as a thread unruffled by the other parts of the framework. NetworkTimer extends *TimerTask* provided by the J2ME API. *TimerTask* requires the method `run()`, and it is this method that gets called each time the timer reaches its interval.
NetworkTimer has a reference to *Network*, and `run()` calls on the defined abstract method `ping()` in *Network*. This will ensure that no exceptions can occur. Listing 16.7 shows how the timer and ping is activated in the class *Bluetooth*.

Listing 16.7: *How NetworkTimer is created, and the ping-functionality is started*

```
protected void activatePing()
{
 if(!pingActivated)
 {
  new NetworkTimer(this,PINGRATE);
  pingActivated = true;
 }
}
```

The ping/echo tactic is described in Section 12.3.4.

16.10.4 Class Diagram

This section shows the class diagram of the network package in Figure 16.15.

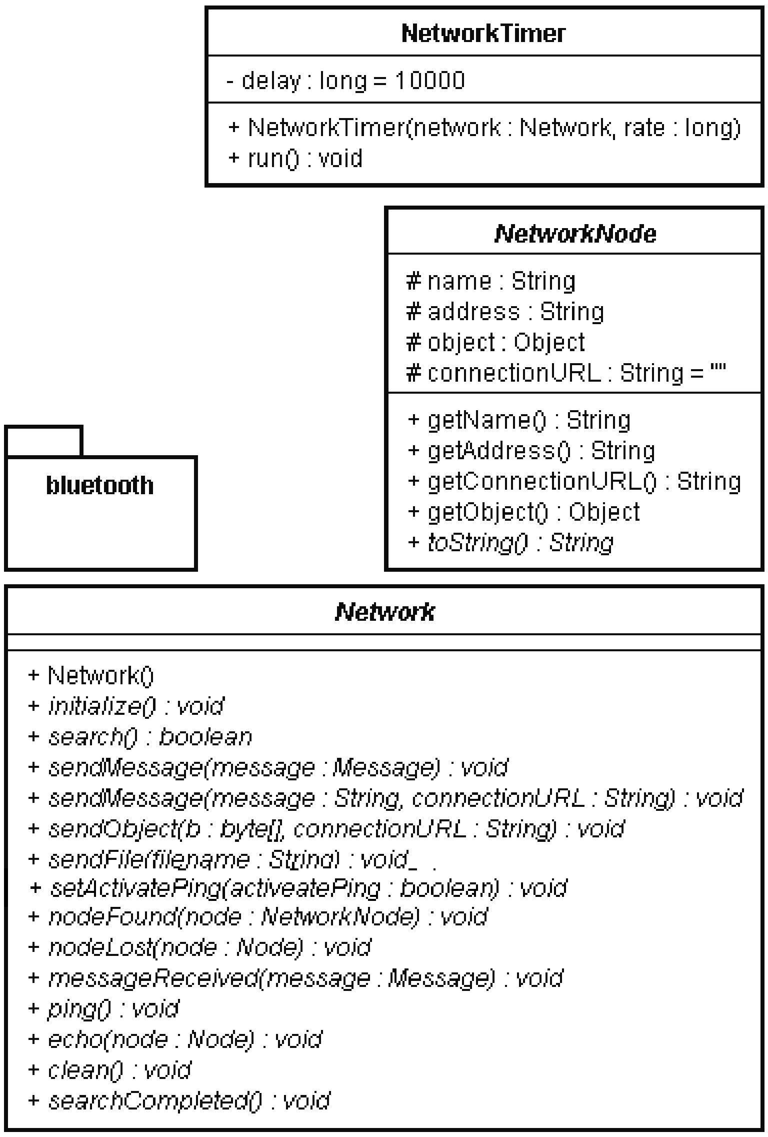

NetworkTimer
- delay : long = 10000
+ NetworkTimer(network : Network, rate : long)
+ run() : void
NetworkNode
# name : String
# address : String
# object : Object
# connectionURL : String = ""
+ getName() : String
+ getAddress() : String
+ getConnectionURL() : String
+ getObject() : Object
+ toString() : String
bluetooth
Network
+ Network()
+ initialize() : void
+ search() : boolean
+ sendMessage(message : Message) : void
+ sendMessage(message : String, connectionURL : String) : void
+ sendObject(b : byte[], connectionURL : String) : void
+ sendFile(filename : String) : void
+ setActivatePing(activeatePing : boolean) : void
+ nodeFound(node : NetworkNode) : void
+ nodeLost(node : Node) : void
+ messageReceived(message : Message) : void
+ ping() : void
+ echo(node : Node) : void
+ clean() : void
+ searchCompleted() : void

Figure 16.15: *Class Diagram of the Network Package*

16.11 The Bluetooth package

This package contains all the classes that are needed to communicate with the Bluetooth network technology. This package extends and implements the classes and interfaces found in the network package. If a new network technology is implemented in the Peer2Me framework, it should offer the same functionality as this package does. The Bluetooth package consists of 12 classes. All the classes and their purposes are described in this section. There are currently two ways of implementing Bluetooth in J2ME: OBEX or RFCOMM. See Section 11.3.1 for more information about these two protocols. The Bluetooth implementation in this version of Peer2Me uses the OBEX protocol for communication. This has great impact on the architecture within this package.

16.11.1 Class: Bluetooth

Extends: peer2me.network.Network

The *Bluetooth* class is the central class in this package and almost everything is indirectly controlled from this class. It is responsible for opening up connections so that other devices may discover the node and connect to it. The message and ping queues are stored in this class and the class is also responsible for creating queue processors for the two queues, see Section 16.12.11 and 16.12.12. All messages must pass through this class, both messages that are to be sent and messages that are received. When messages are received, this class alerts *Session* of the received message. This class also receives instructions by *Session* to send messages. Most of the functionality have been delegated to the other classes in this package. This was necessary to avoid an overly complex network class.

16.11.2 Class: BluetoothListener

Implements: javax.lang.Runnable

This class creates and opens up a connection so that other nodes may find this node and connect to it. This is called creating a local service. In order to create a service, a connection url is needed. A typical connection URL looks like this:

btgoep://localhost:54657374417070546573744170705465;name=TestApp;authenticate=false;encrypt=false

The url consists of different elements:

btgoep The protocol that is used. This url creates an Bluetooth OBEX connection

localhost:### The string requires an UID. The numbers are the UID created by the *HexBuilder* class located in the *peer2me.service* package, see Section 16.7.2. The characters are generated based on the name of the application running the framework.

name=TestApp The name of the running service. This enables similar applications to identify each other so that they can create a connection.

authenticate=false This means that connections do not need to be approved by the node running the service if another node wants to connect. In other words, all connections are accepted.

encrypt=false The data that is exchanged is not encrypted.

When a remote node discovers another node running this service, a connection does not need to be established. However, when a remote node wants to send information to this node, a connection is established and an instance of the *BluetoothConnectionHandler* class is created. As long as one connection has been established, this class will not accept new connections until the current one has finished. The *BluetoothConnectionHandler* instance is dedicated to handle the specific connection. The connection exists until an entire message has been transferred successfully. Then the *BluetoothListener* class is alerted and starts accepting new connections again.

16.11.3 Class: BluetoothConnectionHandler

Extends: javax.obex.ServerRequestHandler

This class is responsible for handling all incoming connections and messages except for pings. Pings are handled by the *BluetoothPingConnectionHandler* class, see Section 16.11.7. When a remote node wants to establish a connection to this node, this class has to accept it. Since OBEX has been used as the protocol for Bluetooth communication, it is very easy to distinguish between different messages. OBEX makes it possible to attach a set of headers to an incoming stream. These headers are described in the *Peer2MeHeaderSet* class, see Section 16.11.11. As mentioned in Section 11.3.1, the OBEX protocol consists of eight operations. This class handles the incoming CONNECT, PUT and DISCONNECT operations. When the `onPut(Operation op)` is invoked in this class, it receives both headers and the stream of bytes. The method first extracts the headers and then reads the stream. Based on the information given in the headers, the stream is read accordingly. The current implementation distinguish between the following type of streams:

- Information about new nodes in the network
- Contents of a file
- Information describing that a node is moved from one group to another
- Primitive variables or serialized object

This class reconstructs the message as data is being received. If only a part of a message is received in a stream, it stores the partial message in a hashtable. When all the parts of a message have been received, the message is sent upwards in the architectural layers, ending up in the MIDlet via the implemented `messageReceived(Message message)` method. Then the message can be handled as illustrated in Listings 16.11.

Listing 16.8: *Handling a received message in the MIDlet*

```java
public void messageReceived(Message message)
{
 boolean containsFile = message.getBoolean("containFiles");
 if(containsFile)
 {
  FileInfo fileInfo = message.getFile("file1");
  String fileName = fileInfo.getFileName();
  String path = fileInfo.getFullPathToFile();
  long size = fileInfo.getFileSize();
 }
 int meaningOfLife = message.getInt("meaning_of_life");
 String plainText = message.getString("string");
 /*
  * <Some code that handles the received information>
  * */
}
```

16.11.4 Class: BluetoothNode

Extends: peer2me.network.NetworkNode

When a reference to a node that is connectable by Bluetooth is needed, an instance of this class is created. This class has one more local variable in addition to the ones inherited from the NetworkNode class called *pingConnectionURL*. The current implementation of Peer2Me creates two ways of reaching another Bluetooth node, either by the ”normal” connection url that is used for message transferring or the other one which is used to see if the node is reachable. There are three constructors in this class, and all are used by the framework. This is because the following three situations occur in the framework regularly:

- The framework needs to create an instance of BluetoothNode of the local node for different purposes.
- The framework needs to create an instance of BluetoothNode of a remote node after it has been found during a regular service search.
- The framework needs to create an instance of BluetoothNode of a remote node A based on information received from a remote node B.

16.11.5 Class: BluetoothObjectPush

This class takes care of connecting and sending messages to a node. A new instance of this class is created for each message that needs to be sent. This class has a method called `sendMessage(Message message)` which will connect and send a message to the recipients of the message. The message will be sent to the recipients *sequentially, not simultaneously.* The proper headers will be set based on the information contained in the message object. This class send the headers to the receiver before the actual stream each time.

16.11.6 Class: BluetoothPingListener

This class has the same purpose as the *BluetoothListener* class, see Section 16.11.2, except that it creates a service that is used for ping-echo purposes only. No messages are transferred using this service. The connection url that this class creates is the same as the other listener class, except for the name of the service.

16.11.7 Class BluetoothPingConnectionHandler

Extends: javax.obex.ServerRequestHandler

This class is similar to the *BluetoothConnectionHandler* class described in Section 16.11.3, but has less functionality. The only purpose of this class is to accept a connection. When a remote node has established a connection to the local node, it disconnects immediately.

16.11.8 Class BluetoothSearcher

Extends: java.lang.Thread
Implements: javax.bluetooth.DiscoveryListener

This class runs in its own thread and extends a required listener, *DiscoveryListener*, defined in the Java Bluetooth API (JSR82 82). The purpose of this class is to search for nearby Bluetooth devices that runs the same application as the local node. The class first finds **all**

Bluetooth devices in vicinity: computers, laptops, mobile phones etc. It then filters and performs a service search on all the mobile phones. If it finds a service on a mobile phone that is similar to the local running service, it creates a reference to the node as a *BluetoothNode* so that a connection can be established later on. After the search has completed it shares the findings with all the nodes that it has created a reference to. In Figure 16.16, node A performs a discovery search and then a service search and finds out that node B and C are running the same service as itself.

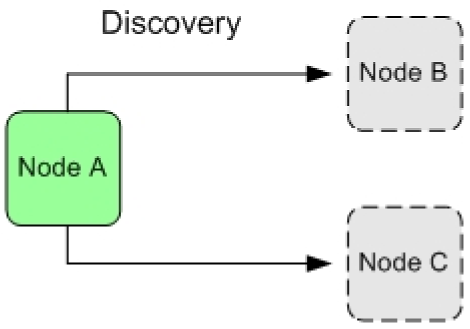


Figure 16.16: *Node A searches and finds B and C*

In Figure 16.17, node A sends messages to node B and C which contains information about all the nodes in the vicinity, which in this case is node A, B and C. Node B, then finds out about A and C. Node C finds out about A and B.

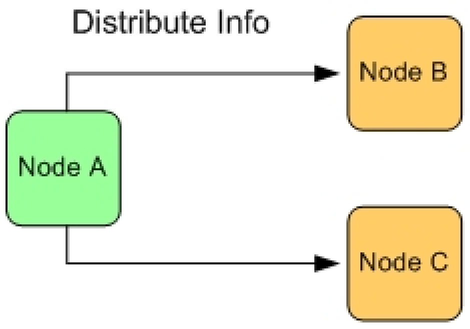


Figure 16.17: *Node A distributes findings*

Figure 16.18 shows the possible connections that can be created by all the nodes. Every node can now connect to everyone. Remember that even though node B and C has not performed any searching, they can still connect to everyone thanks to the discovery done by node A.

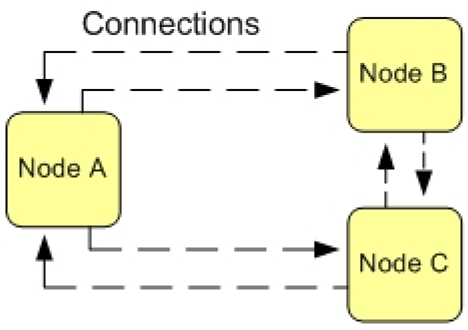


Figure 16.18: *All nodes can now establish connections with each other*

16.11.9 Class: InstanceOfRemoteDevice

Extends: javax.bluetooth.RemoteDevice

This class enables the framework to create an instance of a remote Bluetooth node based on the address of the node. It is not possible to create a *javax.bluetooth.RemoteDevice* based on an address, and therefore this class is necessary. Regularly when a node performs a search, the API returns a *RemoteDevice*, but the Peer2Me framework has now the ability to create a RemoteDevice on its own.

16.11.10 Class: MessageQueueProcessor

Implements: javax.lang.Runnable

This class takes care of processing the messages that has been put in a queue. The queue itself is stored in the *Bluetooth* class. The queue that it processes, is an instance of the class *LinkedMessageList*, see Section 16.12.11. This class runs in its own thread so that the MIDlet does not freeze while the message queue is being processed. In order to send a message, this class creates an instance of *BluetoothObjectPush* in an inner thread so that a message can be aborted if it takes to long.

16.11.11 Class: Peer2MeHeaderSet

Implements: javax.obex.HeaderSet

The OBEX API given in JSR82 contains a class with a standard set of headers that *Peer2HeaderSet* implements. This enables the framework to seamlessly integrate the predefined headers together with custom headers. When a message is sent, it is decomposed into headers and a stream of bytes. Figure 16.19 illustrates the default headers that are sent with every stream of bytes.

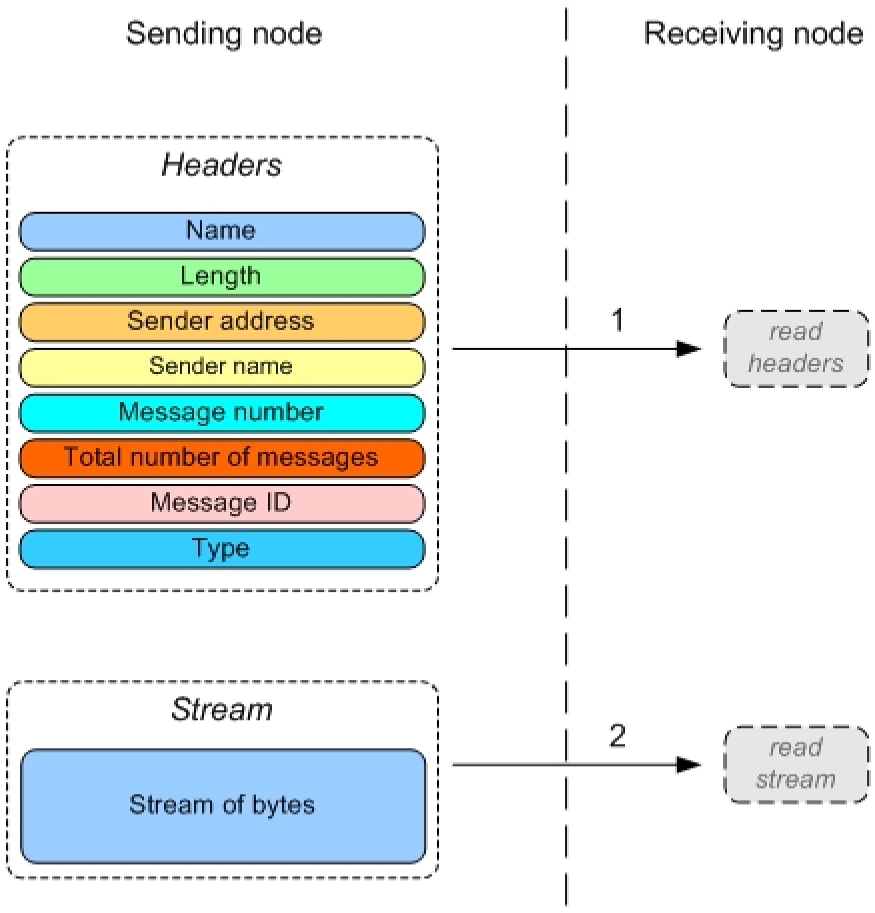


Figure 16.19: *How messages are sent from one node to another.*

In the Javadoc for the *javax.obex.HeaderSet*, an explanation of how to create custom headers is given. When creating custom headers, it is very important to follow the "rules" as shown in Table 16.1.

Header Identifier	**Decimal Range**	**OBEX Type**	**Java Type**
0x30 to 0x3F	48 to 63	Unicode String	java.lang.String
0x70 to 0x7F	112 to 127	byte sequence	byte[]
0xB0 to 0xBF	176 to 191	1 byte	java.lang.Byte
0xF0 to 0xFF	240 to 255	4 byte unsigned integer	java.lang.Long

Table 16.1: *Available user defined OBEX headers. Shows ranges and types, [41]*

OBEX allows 64 user-defined header values.

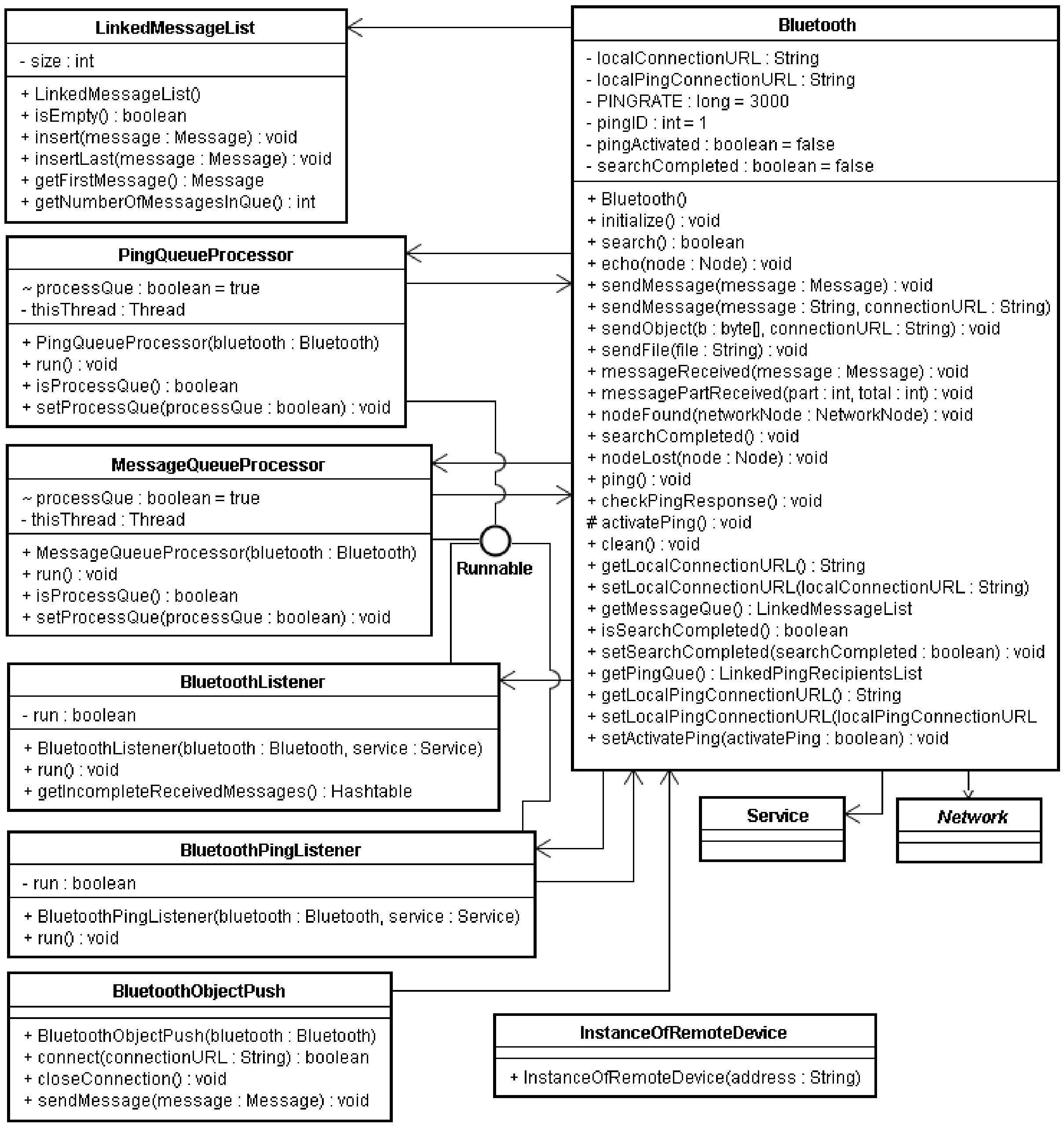
16.11.12 Class: PingQueueProcessor

Implements: javax.lang.Runnable

If the ping-functionality is enabled in the framework, pings are sent with regular intervals to all known nodes. All pings are added to a queue which is stored in the *Bluetooth* class. This class processes the pings in the same way as *MessageQueueProcessor* does. The difference is that this class just tries to connect to the nodes in the queue and reports if the connection was successful. If a ping fails a given number of times, defined in *Node*, in a row, the node is considered lost.

16.11.13 Class Diagrams

This section show the class diagrams of the Bluetooth package in Figure 16.20, Figure 16.21 and 16.22



LinkedMessageList
- size : int
+ LinkedMessageList()
+ isEmpty() : boolean
+ insert(message : Message) : void
+ insertLast(message : Message) : void
+ getFirstMessage() : Message
+ getNumberOfMessagesInQue() : int
PingQueueProcessor
~ processQue : boolean = true
- thisThread : Thread
+ PingQueueProcessor(bluetooth : Bluetooth)
+ run() : void
+ isProcessQue() : boolean
+ setProcessQue(processQue : boolean) : void
MessageQueueProcessor
~ processQue : boolean = true
- thisThread : Thread
+ MessageQueueProcessor(bluetooth : Bluetooth)
+ run() : void
+ isProcessQue() : boolean
+ setProcessQue(processQue : boolean) : void
Runnable
BluetoothListener
- run : boolean
+ BluetoothListener(bluetooth : Bluetooth, service : Service)
+ run() : void
+ getIncompleteReceivedMessages() : Hashtable
BluetoothPingListener
- run : boolean
+ BluetoothPingListener(bluetooth : Bluetooth, service : Service)
+ run() : void
BluetoothObjectPush
+ BluetoothObjectPush(bluetooth : Bluetooth)
+ connect(connectionURL : String) : boolean
+ closeConnection() : void
+ sendMessage(message : Message) : void
InstanceOfRemoteDevice
+ InstanceOfRemoteDevice(address : String)
Bluetooth
- localConnectionURL : String
- localPingConnectionURL : String
- PINGRATE : long = 3000
- pingID : int = 1
- pingActivated : boolean = false
- searchCompleted : boolean = false
+ Bluetooth()
+ initialize() : void
+ search() : boolean
+ echo(node : Node) : void
+ sendMessage(message : Message) : void
+ sendMessage(message : String, connectionURL : String)
+ sendObject(b : byte[], connectionURL : String) : void
+ sendFile(file : String) : void
+ messageReceived(message : Message) : void
+ messagePartReceived(part : int, total : int) : void
+ nodeFound(networkNode : NetworkNode) : void
+ searchCompleted() : void
+ nodeLost(node : Node) : void
+ ping() : void
+ checkPingResponse() : void
# activatePing() : void
+ clean() : void
+ getLocalConnectionURL() : String
+ setLocalConnectionURL(localConnectionURL : String)
+ getMessageQue() : LinkedMessageList
+ isSearchCompleted() : boolean
+ setSearchCompleted(searchCompleted : boolean) : void
+ getPingQue() : LinkedPingRecipientsList
+ getLocalPingConnectionURL() : String
+ setLocalPingConnectionURL(localPingConnectionURL
+ setActivatePing(activatePing : boolean) : void
Service
Network

Figure 16.20: *Class Diagram of the Bluetooth Package, part 1*

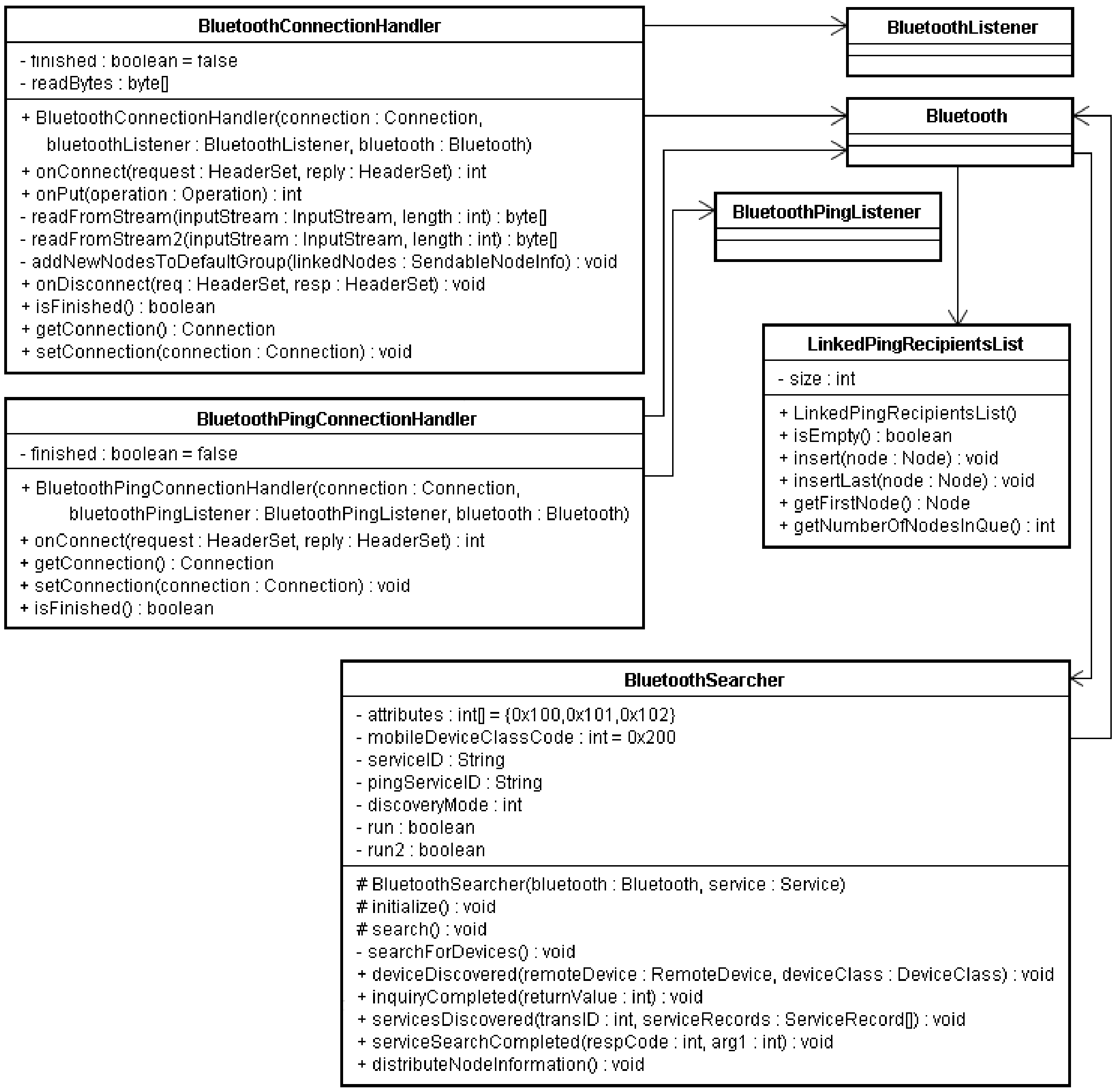

BluetoothConnectionHandler
- finished : boolean = false
- readBytes : byte[]
+ BluetoothConnectionHandler(connection : Connection, bluetoothListener : BluetoothListener, bluetooth : Bluetooth)
+ onConnect(request : HeaderSet, reply : HeaderSet) : int
+ onPut(operation : Operation) : int
- readFromStream(inputStream : InputStream, length : int) : byte[]
- readFromStream2(inputStream : InputStream, length : int) : byte[]
- addNewNodesToDefaultGroup(linkedNodes : SendableNodeInfo) : void
+ onDisconnect(req : HeaderSet, resp : HeaderSet) : void
+ isFinished() : boolean
+ getConnection() : Connection
+ setConnection(connection : Connection) : void
BluetoothListener
Bluetooth
BluetoothPingListener
BluetoothPingConnectionHandler
- finished : boolean = false
+ BluetoothPingConnectionHandler(connection : Connection, bluetoothPingListener : BluetoothPingListener, bluetooth : Bluetooth)
+ onConnect(request : HeaderSet, reply : HeaderSet) : int
+ getConnection() : Connection
+ setConnection(connection : Connection) : void
+ isFinished() : boolean
LinkedPingRecipientsList
- size : int
+ LinkedPingRecipientsList()
+ isEmpty() : boolean
+ insert(node : Node) : void
+ insertLast(node : Node) : void
+ getFirstNode() : Node
+ getNumberOfNodesInQue() : int
BluetoothSearcher
- attributes : int[] = {0x100,0x101,0x102}
- mobileDeviceClassCode : int = 0x200
- serviceID : String
- pingServiceID : String
- discoveryMode : int
- run : boolean
- run2 : boolean
# BluetoothSearcher(bluetooth : Bluetooth, service : Service)
# initialize() : void
# search() : void
- searchForDevices() : void
+ deviceDiscovered(remoteDevice : RemoteDevice, deviceClass : DeviceClass) : void
+ inquiryCompleted(returnValue : int) : void
+ servicesDiscovered(transID : int, serviceRecords : ServiceRecord[]) : void
+ serviceSearchCompleted(respCode : int, arg1 : int) : void
+ distributeNodeInformation() : void

Figure 16.21: *Class Diagram of the Bluetooth Package, part 2*

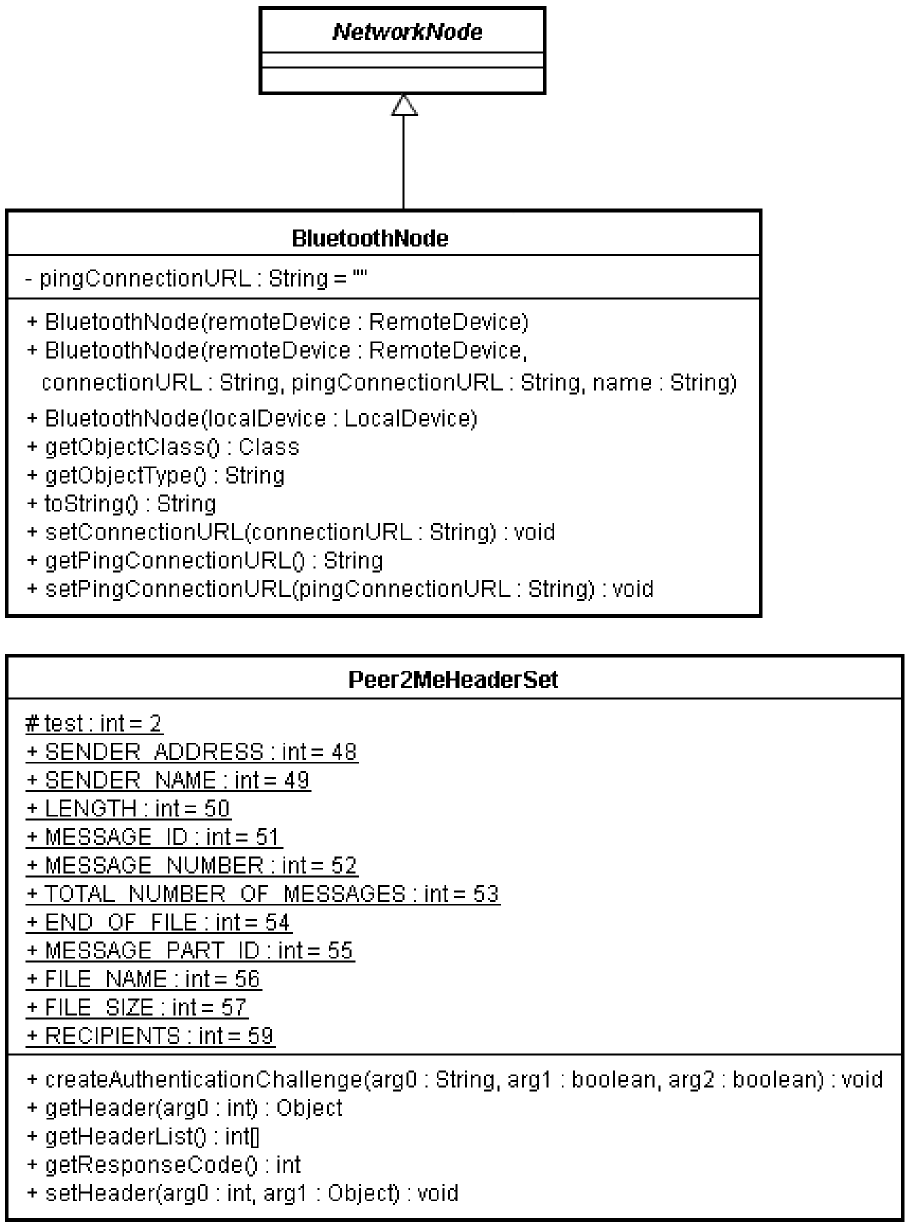

NetworkNode
BluetoothNode
- pingConnectionURL : String = ""
+ BluetoothNode(remoteDevice : RemoteDevice)
+ BluetoothNode(remoteDevice : RemoteDevice, connectionURL : String, pingConnectionURL : String, name : String)
+ BluetoothNode(localDevice : LocalDevice)
+ getObjectClass() : Class
+ getObjectType() : String
+ toString() : String
+ setConnectionURL(connectionURL : String) : void
+ getPingConnectionURL() : String
+ setPingConnectionURL(pingConnectionURL : String) : void
Peer2MeHeaderSet
# test : int = 2
+ SENDER_ADDRESS : int = 48
+ SENDER_NAME : int = 49
+ LENGTH : int = 50
+ MESSAGE_ID : int = 51
+ MESSAGE_NUMBER : int = 52
+ TOTAL_NUMBER_OF_MESSAGES : int = 53
+ END_OF_FILE : int = 54
+ MESSAGE_PART_ID : int = 55
+ FILE_NAME : int = 56
+ FILE_SIZE : int = 57
+ RECIPIENTS : int = 59
+ createAuthenticationChallenge(arg0 : String, arg1 : boolean, arg2 : boolean) : void
+ getHeader(arg0 : int) : Object
+ getHeaderList() : int[]
+ getResponseCode() : int
+ setHeader(arg0 : int, arg1 : Object) : void

Figure 16.22: *Class Diagram of the Bluetooth Package, part 3*

16.12 The Message Package

The Peer2Me framework is able to send and receive complex messages and this package contains all the classes that are needed to do this. The message package fulfills the functional requirements 5 - 10 that are described in Table 14.1. These requirements are far beyond the implementation of message handling in the old Peer2Me framework. The package consists of 13 classes. The classes are all connected, but some classes are more semantically coherent than others. This is presented in Figure 16.23.

All the classes and their purposes are described in this section. However, a brief overview will now be given.

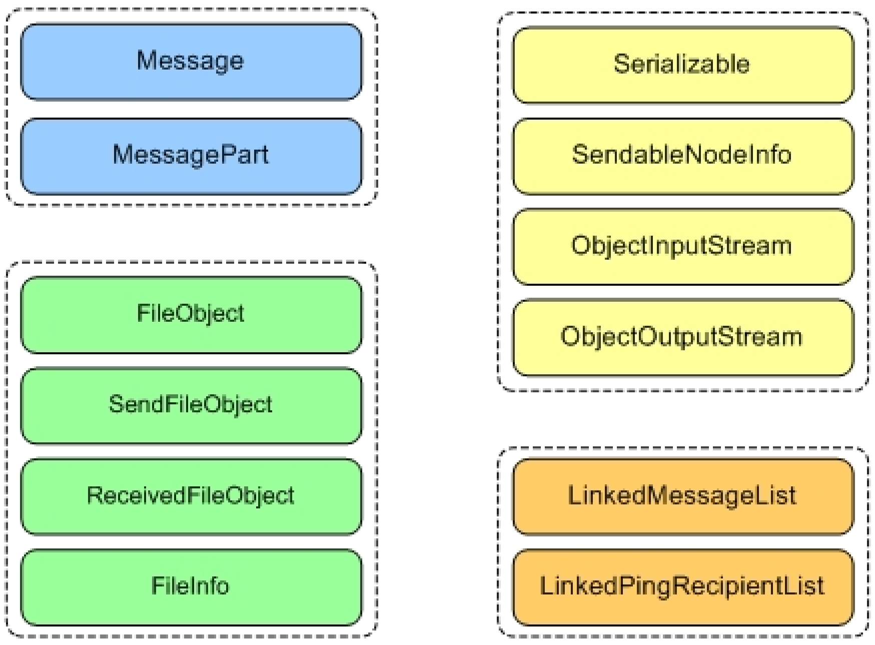


Figure 16.23: *The classes in the Message package*

The *Message* class may contain many *MessageParts*. A *MessagePart* can contain a child-class of *FileObject*. The classes *SendFileObject* and *ReceivedFileObject* both extend the class *FileObject*. *FileInfo* is a class that contains information about a received file. The framework is also able to send serialized objects. These objects must be instances of classes that implements the *Serializable*[3] interface. *SendableNodeInfo* is such a class. The classes *ObjectInputStream* and *ObjectOutPutStream* are used to add and retrieve serialized objects from a *Message*. LinkedMessageList is a custom dynamic list that is used to add Messages to a queue in the framework. The handling of the queue is done in another class, *MessageQueueProcessor*. *LinkedPingRecipientList* is also a custom dynamic list that contains information about which nodes that must be pinged. This queue is handled by *PingQueueProcessor*.

16.12.1 Class: Message

In order to send a message from one node in the network to another, an instance of the Message class must be created. The Message class can contain different types of information. There are three different kinds of information that can be attached to a Message, illustrated in Figure 16.24. The elements in the figure is explained in more detail in Section

Creating Messages and Sending Them

Messages can be created and sent in a much easier fashion than in the old version. This can be illustrated by a few code examples. Listing 16.9 shows how to create and send a simple message in the old framework.

[3]Serialize: In order to represent an instance of an object as pute bytes, the class must implement a Serializable interface. The API for J2SE and J2EE has such an interface, but the API for J2ME does not.

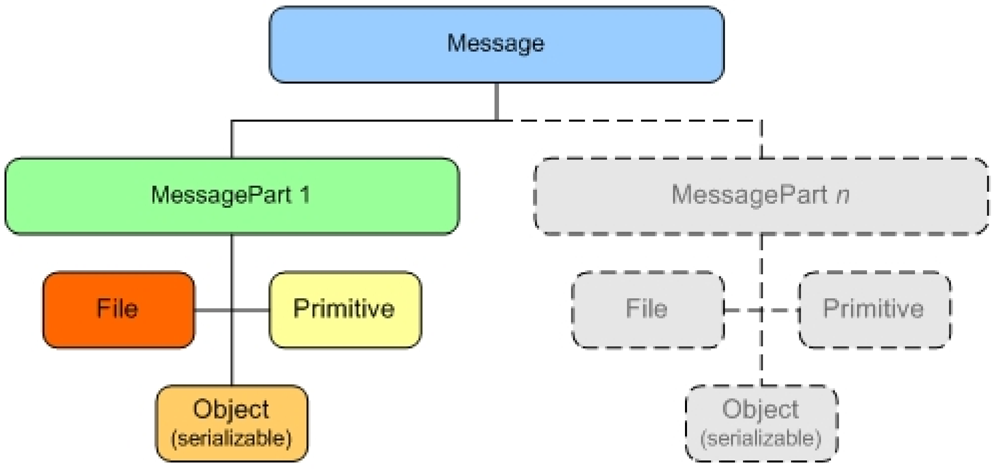


Figure 16.24: *Possible contents of a Message*

Listing 16.9: *Example of creating and sending a simple message in the old framework*

```
Message message = new Message();
TextMessagePart text = new TextMessagePart();
text.setDescription("example");
text.setFieldValue("Sample text");
message.addMessageBodyPart(text);
message.addRecipient(node);
framework.sendMessage(message,service);
```

Listing 16.10 shows the new way of creating and sending a simple message.

Listing 16.10: *Example of creating and sending a simple message in the new framework*

```
Message message = new Message();
message.addElement("Sample text","example");
message.addRecipient(node);
framework.sendMessage(message);
```

In the old framework, a new `MessagePart` had to be created for each `String` you wanted to attach to a `Message`. This resulted in 4 lines of code. In the new framework, this has been reduced to 1 line of code; message.addElement(”Sample text”,”example”). The second parameter in the `addElement` method is the key that is associated with the text string. The developer does not even have to worry about creating instances of `MessagePart` objects and attaching those to the message. The new framework still uses the concept of messageparts, but hides them from the application developer and only uses them internally in the framework. The new Peer2Me also allows the developer to add other primitive datatypes to the message and assign a ”key” to each of them. Listings 16.11 shows a Message in the new framework that contains a String, an int, a double, a boolean, a serialized object and two files.

Listing 16.11: *Example of creating and sending a complex message in the new framework*

```
Message message = new Message();
message.addElement("This message contains plenty of stuff","string");
message.addElement(42,"meaning_of_life");
message.addElement(3.14,"pi");
message.addElement(true,"containFiles");
message.addSerializedObject(customObject, "myObject");
message.addFile("file1", "e:/MSSEMC/Media files/audio/Crazy.mp3");
message.addFile("file2", "e:/MSSEMC/Media files/audio/funny.mp3");
message.addRecipient(node);
framework.sendMessage(message);
```

Important methods

All methods in the class are used in the framework and documented with Javadoc, but there are a few that needs further explanation. It is not expected that the reader understands the complete workings of these methods, but it is very important for a developer that wants to modify the framework to be aware of the importance of these methods.

storeAllPrimitivesInOnePart() - When primitive variables and/or Strings are added to an instance of *Message*, they are stored in separate instances of *MessagePart*s. When the framework is instructed to send the message, these variables are stored in **one** part, so that all primitive values can be converted into a stream of bytes that can be sent over the network.

extractPrimitivesToSeparateParts() - The receiving node must extract all the primitives back to their original *MessagePart*s.

numberOfChunksNeeded() - The number of parts in a message determines how many "chunks" of bytestreams the message must be split in before it can be sent over the network. Keep in mind that all primitive values have been stored in one part prior to sending and if a message only contains primitive values, only one chunk is needed. If three serialized objects are added in addition to the primitive values, four chunks are needed. If a file is added, the number of chunks needed is determined by the size of the file and how large the chunks of the file should be sent each time. The size is set programmatically in the *SendFileObject* class.

addFile(String key, String fullPath) - This method must be used by the application developer when adding files. A call to this method, will add a part containing an instance of *SendFileObject* to the message. The contents of the file will not be read until the message is sent.

addReceivedFile(String key, FileObject fileObject) - The node that receives a message containing a file, will add a part containing a *ReceivedFileObject* to the message.

addFileInfo(String key, FileInfo fileInfo) - After all the chunks of a message have been received, an instance of *FileInfo* will be added to the message. This object should be used if the receiving node wants details about the file it just received. The object is retrieved by calling the method `getFileInfo(String key)` on the received message object.

16.12.2 Class: MessagePart

As mentioned in the previous section, a Message can contain multiple MessageParts. A MessagePart can only contain one of the following elements:

- A file
- A serialized object
- A primitive data type or a String

The class has one constructor for each type of MessagePart that can be created. There are 12 constructors in MessagePart. There are also get-methods for each of the elements. For instance if a MessagePart is created using this constructor: `MessagePart(String key, double doubleValue)`, a call to the method `getDoubleValue()` will return the double value that was given in the constructor. When creating an instance of MessagePart, the class automatically knows what kind of information it contains by the local variable, `type`. This enables the framework to easily determine what kind of information is stored in a MessagePart.

The MessagePart class exposes public variables that must be used to determine the type of message. The reader is invited to inspect the Javadoc and the source code for a more detailed view of the logics of the class.

16.12.3 Abstract Class: FileObject

This class is an abstract class. There are two other classes that extends this class: *SendFileObject* and *ReceivedFileObject*. The abstract class contains variables and methods that are common for the to child classes such as filename, file size, full path to the file and if access has been granted to the file.

16.12.4 Class: SendFileObject

Extends: peer2me.message.FileObject
Implements: java.lang.Runnable

This class extends the *FileObject* class. If a node wants to attach a file to a message and send it over the network, an instance of this class is created and added to the message by the framework. When the framework is instructed to send the message, a connection to the file is established. This connection **must** be approved by the user of the application. The display on the mobile phone will show the following messages: ”Allow application to read user data? / Allow application to write user data?”. The user must answer ”yes” to these questions to allow access to the filesystem. After this is done, this class can begin reading from the file.

16.12.5 Class: ReceivedFileObject

Extends: peer2me.message.FileObject
Implements: java.lang.Runnable

This class extends the *FileObject* class. When a node receives a message containing a file, an instance of this class is created and added to the message by the framework. Before the file is created, the user will be asked to allow the application to write user data. When the user confirms this, the framework creates the file and writes content to it. For each chunk of file that is received, new data is written to the file by this class.

16.12.6 Class: FileInfo

After a node has received an entire file, an instance of the FileInfo class is created and attached to the message. This class contains useful information about the file that has been received such as: filename, path to file and the size of the file. As described in Section 16.12.1, a key is associated with a file. The same key must be used on the receiving node to get information about the file. Recall in Listings 16.11 how a file was added to a message. Listings 16.12 shows how to retrieve information about the file associated with key ”file1” on the receiving node.

Listing 16.12: *Retrieving info about a file*

```
FileInfo fileInfo = message.getFile("file1");
String fileName = fileInfo.getFileName();
String path = fileInfo.getFullPathToFile();
long size = fileInfo.getFileSize();
```

For instance, the variable `fileName` will contain the String: ”Crazy.mp3”, while `filePath` will contain the String: ”e:/MSSEMC/Media files/audio/Crazy.mp3”.

16.12.7 Interface: Serializable

In order for the framework to send objects over the network, the objects **must** implement this interface. The interface has two methods: one for serializing and one for deserializing. This interface was created since there are currently no serializable interfaces available for J2ME. Classes that implements this interface must also implement an empty constructor. The Peer2Me framework has one class that implements this interface: *SendableNodeInfo*. Application developers are encouraged to use this interface.

16.12.8 Class: SendableNodeInfo

Implements: peer2me.message.Serializable

This class implements the *Serializable* interface and instances of this class can therefore be sent over the network. This class is used by the framework to send information about all the nodes in the network. This class is used as a linked list, since it also contains an instance of itself. This enables the framework to iterate through the nodes easily. The linked list is created in the *Bluetooth* package and is sent to all the nodes right after a search has completed.

Developers that want to create their own serialized classes should use the *SendableNodeInfo* class as an example.

16.12.9 Class ObjectOutputStream

When a serialized object is added to a message, this class serializes it into a bytestream so that it can be sent over the network. This class calls the implemented `serialize()` method from the *Serializable* interface.

16.12.10 Class ObjectInputStream

When a node receives a stream of bytes, this class deserializes the stream and creates an object of it. This class calls the implemented `deserialize()` method from the *Serializable* interface. The object that is created must be casted to the correct class. This task must be solved by the application developer.

16.12.11 Class LinkedMessageList

This class is used as a message queue by the framework. Each time the `sendMessage(Message message)` is called on the framework, the message is added to this queue. The queue follows the First-In-First-Out (FIFO) algorithm. The algorithm is described in Section 12.3.3. In Figure 16.25, message A is added to the queue first, then B, C and D. When the framework processes the queue, it removes message A from the queue, sets the next message in the queue to B and processes message A.

16.12.12 Class LinkedPingRecipientsList

The framework uses a ping-tactic to increase availability and this class is used as a ”ping-recipient-queue”. The ping-tactic is described in Section 12.3.4. The framework uses this queue in the background and the queue is created and processed according to a timeinterval that is set in the framework. This whole process is not visible for the application developer. The queue follows the same structure and prioritizes pings according to the FIFO algorithm in the same way as the *LinkedMessageList* class described in Section 16.12.11 prioritizes messages.

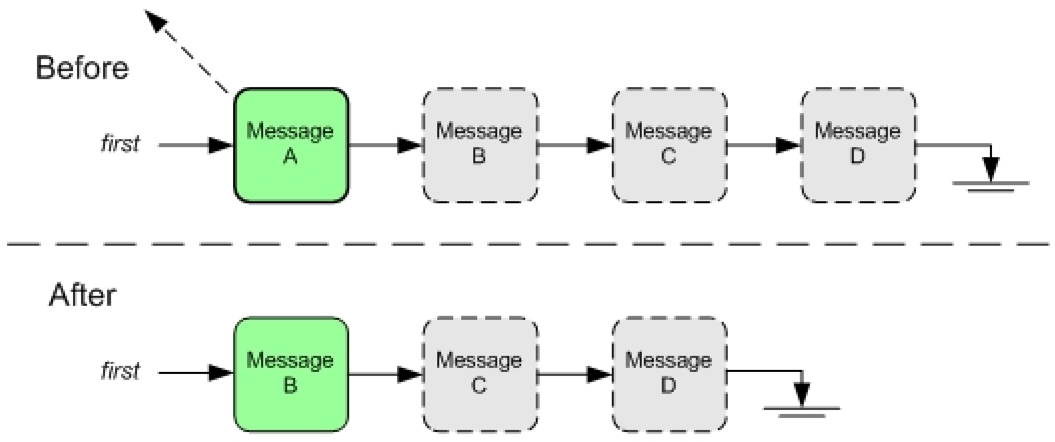


Figure 16.25: *The message queue. Before and after node A has been removed from the queue.*

16.12.13 Class Diagrams

This section shows the class diagrams of the message package in Figure 16.26.

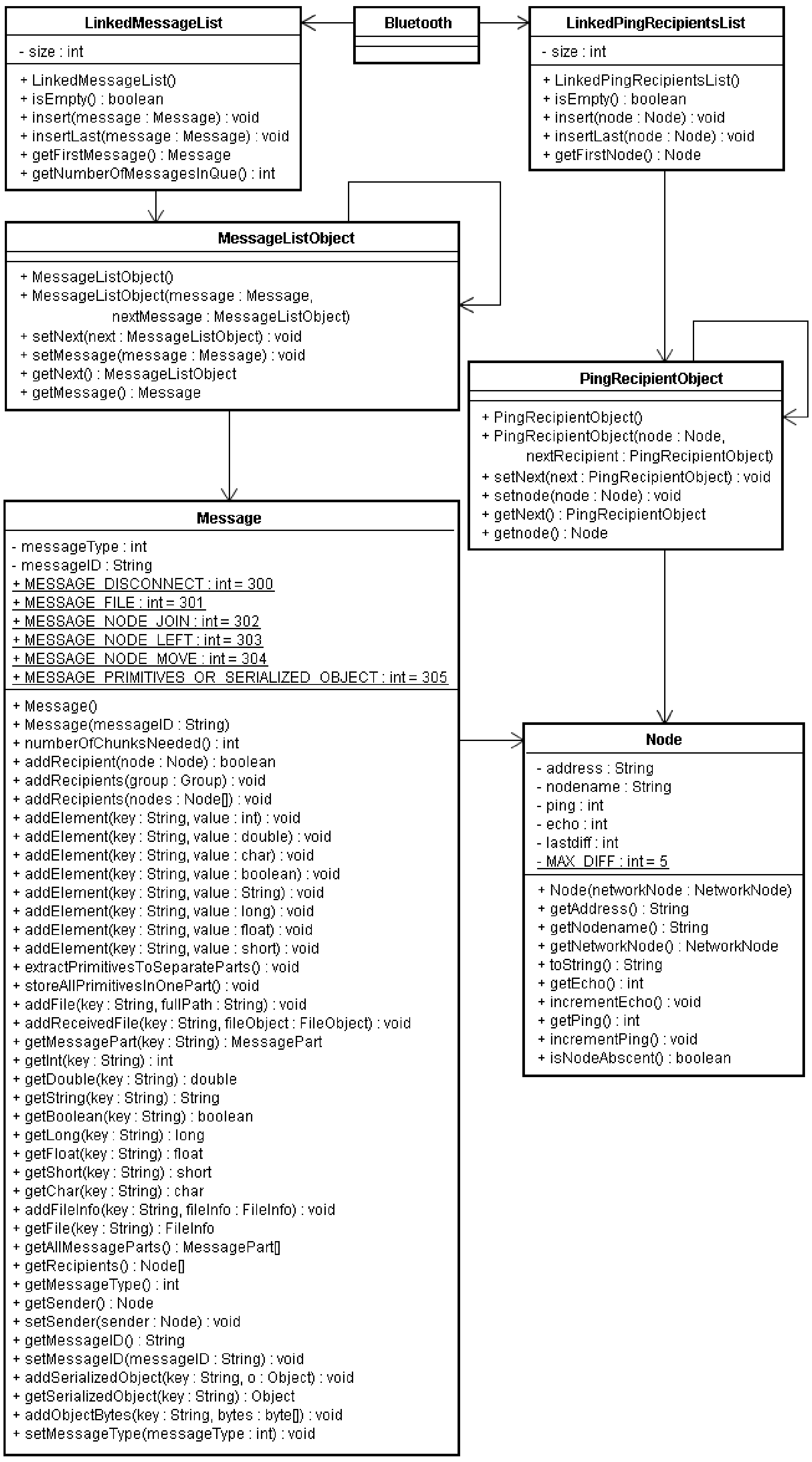

LinkedMessageList
- size : int
+ LinkedMessageList()
+ isEmpty() : boolean
+ insert(message : Message) : void
+ insertLast(message : Message) : void
+ getFirstMessage() : Message
+ getNumberOfMessagesInQue() : int
Bluetooth
LinkedPingRecipientsList
- size : int
+ LinkedPingRecipientsList()
+ isEmpty() : boolean
+ insert(node : Node) : void
+ insertLast(node : Node) : void
+ getFirstNode() : Node
MessageListObject
+ MessageListObject()
+ MessageListObject(message : Message, nextMessage : MessageListObject)
+ setNext(next : MessageListObject) : void
+ setMessage(message : Message) : void
+ getNext() : MessageListObject
+ getMessage() : Message
PingRecipientObject
+ PingRecipientObject()
+ PingRecipientObject(node : Node, nextRecipient : PingRecipientObject)
+ setNext(next : PingRecipientObject) : void
+ setnode(node : Node) : void
+ getNext() : PingRecipientObject
+ getnode() : Node
Message
- messageType : int
- messageID : String
+ MESSAGE_DISCONNECT : int = 300
+ MESSAGE_FILE : int = 301
+ MESSAGE_NODE_JOIN : int = 302
+ MESSAGE_NODE_LEFT : int = 303
+ MESSAGE_NODE_MOVE : int = 304
+ MESSAGE_PRIMITIVES_OR_SERIALIZED_OBJECT : int = 305
+ Message()
+ Message(messageID : String)
+ numberOfChunksNeeded() : int
+ addRecipient(node : Node) : boolean
+ addRecipients(group : Group) : void
+ addRecipients(nodes : Node[]) : void
+ addElement(key : String, value : int) : void
+ addElement(key : String, value : double) : void
+ addElement(key : String, value : char) : void
+ addElement(key : String, value : boolean) : void
+ addElement(key : String, value : String) : void
+ addElement(key : String, value : long) : void
+ addElement(key : String, value : float) : void
+ addElement(key : String, value : short) : void
+ extractPrimitivesToSeparateParts() : void
+ storeAllPrimitivesInOnePart() : void
+ addFile(key : String, fullPath : String) : void
+ addReceivedFile(key : String, fileObject : FileObject) : void
+ getMessagePart(key : String) : MessagePart
+ getInt(key : String) : int
+ getDouble(key : String) : double
+ getString(key : String) : String
+ getBoolean(key : String) : boolean
+ getLong(key : String) : long
+ getFloat(key : String) : float
+ getShort(key : String) : short
+ getChar(key : String) : char
+ addFileInfo(key : String, fileInfo : FileInfo) : void
+ getFile(key : String) : FileInfo
+ getAllMessageParts() : MessagePart[]
+ getRecipients() : Node[]
+ getMessageType() : int
+ getSender() : Node
+ setSender(sender : Node) : void
+ getMessageID() : String
+ setMessageID(messageID : String) : void
+ addSerializedObject(key : String, o : Object) : void
+ getSerializedObject(key : String) : Object
+ addObjectBytes(key : String, bytes : byte[]) : void
+ setMessageType(messageType : int) : void
Node
- address : String
- nodename : String
- ping : int
- echo : int
- lastdiff : int
- MAX_DIFF : int = 5
+ Node(networkNode : NetworkNode)
+ getAddress() : String
+ getNodename() : String
+ getNetworkNode() : NetworkNode
+ toString() : String
+ getEcho() : int
+ incrementEcho() : void
+ getPing() : int
+ incrementPing() : void
+ isNodeAbscent() : boolean

Figure 16.26: *Class Diagram of the Message Package, part 1*

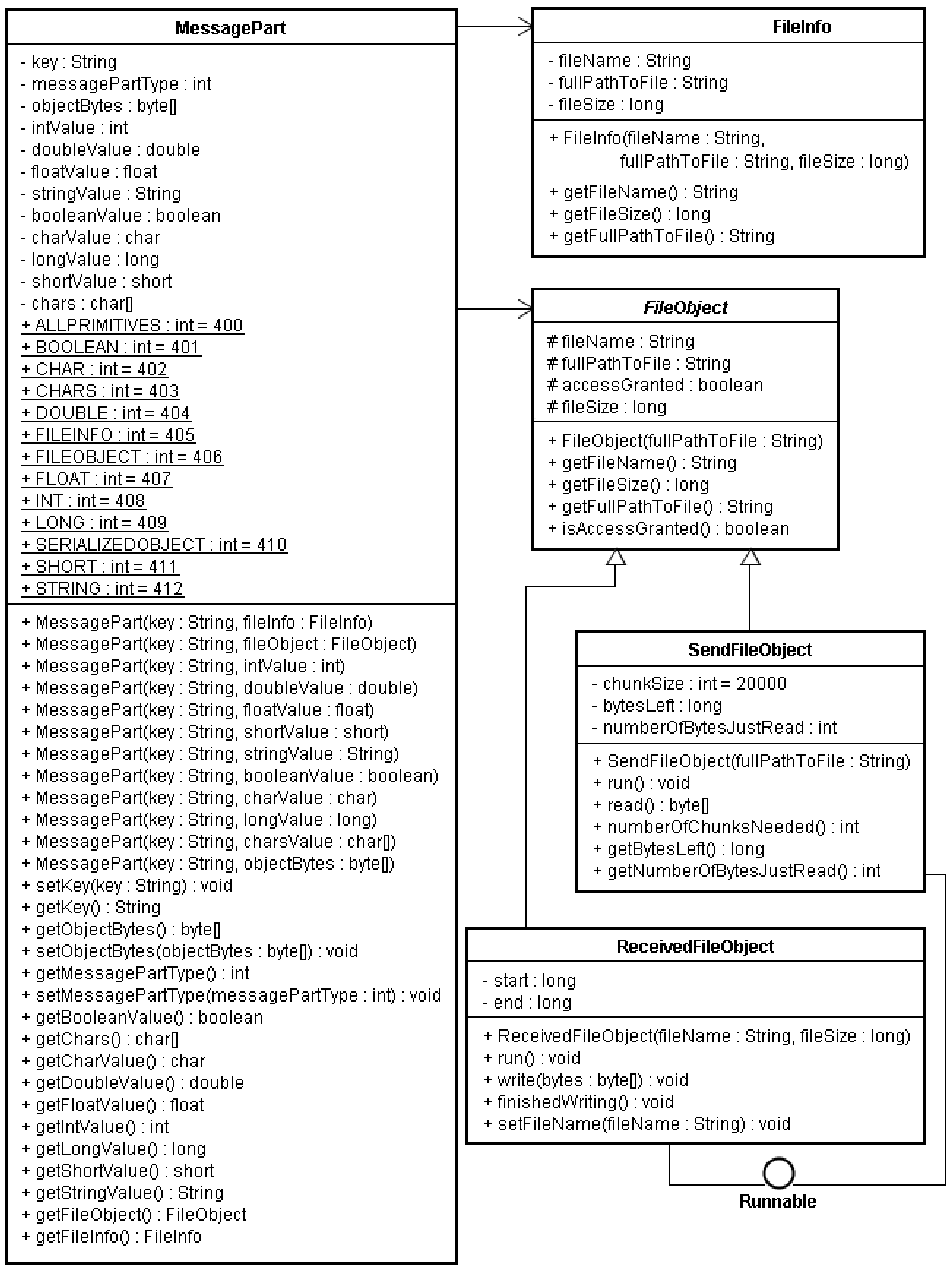

MessagePart
- key : String
- messagePartType : int
- objectBytes : byte[]
- intValue : int
- doubleValue : double
- floatValue : float
- stringValue : String
- booleanValue : boolean
- charValue : char
- longValue : long
- shortValue : short
- chars : char[]
+ ALLPRIMITIVES : int = 400
+ BOOLEAN : int = 401
+ CHAR : int = 402
+ CHARS : int = 403
+ DOUBLE : int = 404
+ FILEINFO : int = 405
+ FILEOBJECT : int = 406
+ FLOAT : int = 407
+ INT : int = 408
+ LONG : int = 409
+ SERIALIZEDOBJECT : int = 410
+ SHORT : int = 411
+ STRING : int = 412
+ MessagePart(key : String, fileInfo : FileInfo)
+ MessagePart(key : String, fileObject : FileObject)
+ MessagePart(key : String, intValue : int)
+ MessagePart(key : String, doubleValue : double)
+ MessagePart(key : String, floatValue : float)
+ MessagePart(key : String, shortValue : short)
+ MessagePart(key : String, stringValue : String)
+ MessagePart(key : String, booleanValue : boolean)
+ MessagePart(key : String, charValue : char)
+ MessagePart(key : String, longValue : long)
+ MessagePart(key : String, charsValue : char[])
+ MessagePart(key : String, objectBytes : byte[])
+ setKey(key : String) : void
+ getKey() : String
+ getObjectBytes() : byte[]
+ setObjectBytes(objectBytes : byte[]) : void
+ getMessagePartType() : int
+ setMessagePartType(messagePartType : int) : void
+ getBooleanValue() : boolean
+ getChars() : char[]
+ getCharValue() : char
+ getDoubleValue() : double
+ getFloatValue() : float
+ getIntValue() : int
+ getLongValue() : long
+ getShortValue() : short
+ getStringValue() : String
+ getFileObject() : FileObject
+ getFileInfo() : FileInfo
FileInfo
- fileName : String
- fullPathToFile : String
- fileSize : long
+ FileInfo(fileName : String, fullPathToFile : String, fileSize : long)
+ getFileName() : String
+ getFileSize() : long
+ getFullPathToFile() : String
FileObject
# fileName : String
# fullPathToFile : String
# accessGranted : boolean
# fileSize : long
+ FileObject(fullPathToFile : String)
+ getFileName() : String
+ getFileSize() : long
+ getFullPathToFile() : String
+ isAccessGranted() : boolean
SendFileObject
- chunkSize : int = 20000
- bytesLeft : long
- numberOfBytesJustRead : int
+ SendFileObject(fullPathToFile : String)
+ run() : void
+ read() : byte[]
+ numberOfChunksNeeded() : int
+ getBytesLeft() : long
+ getNumberOfBytesJustRead() : int
ReceivedFileObject
- start : long
- end : long
+ ReceivedFileObject(fileName : String, fileSize : long)
+ run() : void
+ write(bytes : byte[]) : void
+ finishedWriting() : void
+ setFileName(fileName : String) : void
Runnable

Figure 16.27: *Class Diagram of the Message Package, part 2*

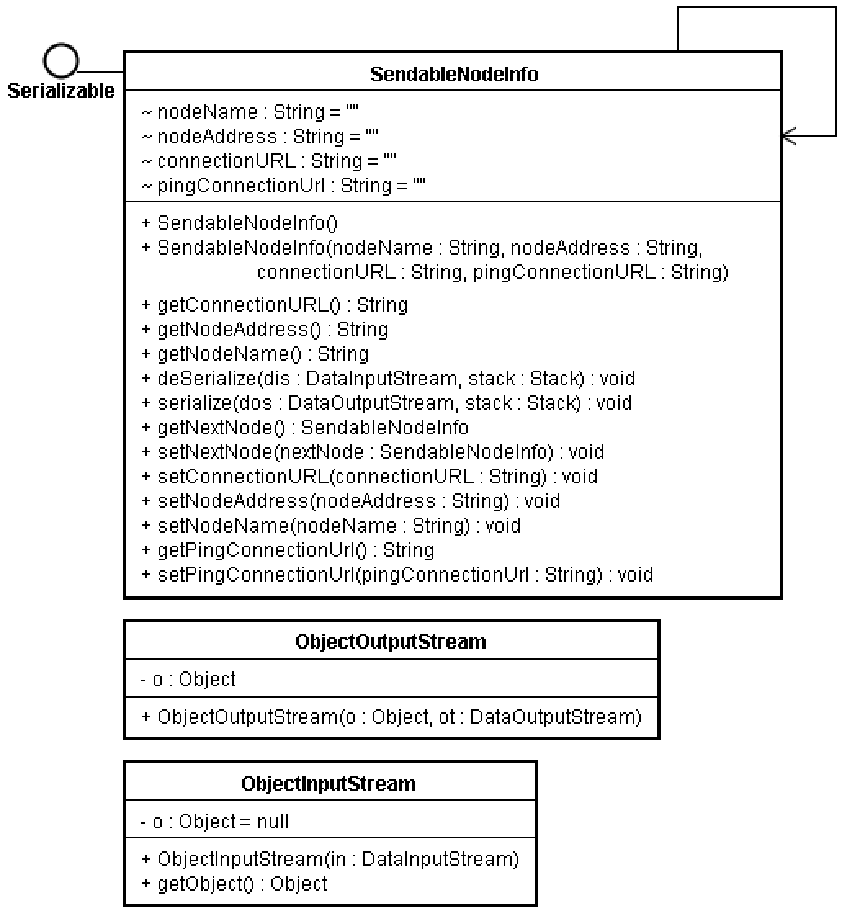

Serializable
SendableNodeInfo
~ nodeName : String = ""
~ nodeAddress : String = ""
~ connectionURL : String = ""
~ pingConnectionUrl : String = ""
+ SendableNodeInfo()
+ SendableNodeInfo(nodeName : String, nodeAddress : String, connectionURL : String, pingConnectionURL : String)
+ getConnectionURL() : String
+ getNodeAddress() : String
+ getNodeName() : String
+ deSerialize(dis : DataInputStream, stack : Stack) : void
+ serialize(dos : DataOutputStream, stack : Stack) : void
+ getNextNode() : SendableNodeInfo
+ setNextNode(nextNode : SendableNodeInfo) : void
+ setConnectionURL(connectionURL : String) : void
+ setNodeAddress(nodeAddress : String) : void
+ setNodeName(nodeName : String) : void
+ getPingConnectionUrl() : String
+ setPingConnectionUrl(pingConnectionUrl : String) : void
ObjectOutputStream
- o : Object
+ ObjectOutputStream(o : Object, ot : DataOutputStream)
ObjectInputStream
- o : Object = null
+ ObjectInputStream(in : DataInputStream)
+ getObject() : Object

Figure 16.28: *Class Diagram of the Message Package, part 3*

16.13 The Exception package

An important thing in all software solutions is handling exceptions and errors. One way is to try/catch most of the code, but this is definitively not a *best practice*. The important thing is to catch and throw the right exceptions giving good explanation of what the exception is about. The typically thing to do is extending a Java Exception, inheriting the base functionality and adding your own. You can then throw your own exception. Figure 16.29 shows the path of inheritance.

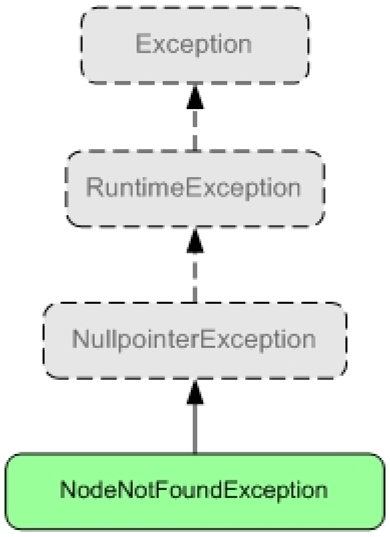


Figure 16.29: *An example of the inheritance of exception*

This version of Peer2Me uses several own exception types, all described briefly in the preceding sections.

16.13.1 FileNotFoundException

Extends: java.lang.NullPointerException

This exception is thrown when the framework tries to read or send a file that not exist.

16.13.2 FrameworkNotInitializedException

Extends: java.lang.NullPointerException

This exception is thrown if the framework is not initialized before search is called. The method `framework.initialize()` must be called before trying to search for devices.

16.13.3 GroupNotFoundException

This exception is thrown when getting or removing a group from the framework. Is thrown by the methods `getGroup(..)` and `removeGroup(..)`. The reason is that the group at the specified index or with the specified name does not exist.
Extends *NullpoinerException*.

16.13.4 NodeNotFoundException

Extends: java.lang.NullPointerException

This exception is thrown when getting, moving or removing a node from the framework. Is thrown by the methods `getNode(..)`, `moveNode(..)` and `removeNode(..)`. The reason is that the node at the specified index and group, or a node with the specified address does not exist.

16.13.5 LocalDeviceNotFoundException

Extends: java.lang.NullPointerException

This exception is thrown when the framework tries get properties from the mobile device, but fails. This can be because of Bluetooth security issues or if the Bluetooth does not function properly.

16.13.6 HexConversionException

Extends: java.lang.Exception

This exception is thrown if the process of converting an *ASCII-string* to a *HEX-string fails*. Will occur if a *ServiceID* in ASCII-characters cannot be converted to a **32-bit** HEX-string.

16.13.7 InvalidKeyException

Extends: java.lang.IllegalArgumentException

This exception is thrown by `addSerializedObject(String key, Object o)` if the key is invalid.

16.13.8 UnknownKeyException

Extends: java.lang.IllegalArgumentException

This exception is thrown by `getSerializedObject(String key)` if the key is invalid, or if the key points to nothing.

16.13.9 Class Diagram

This section show the class diagram of the exception package in Figure 16.30.

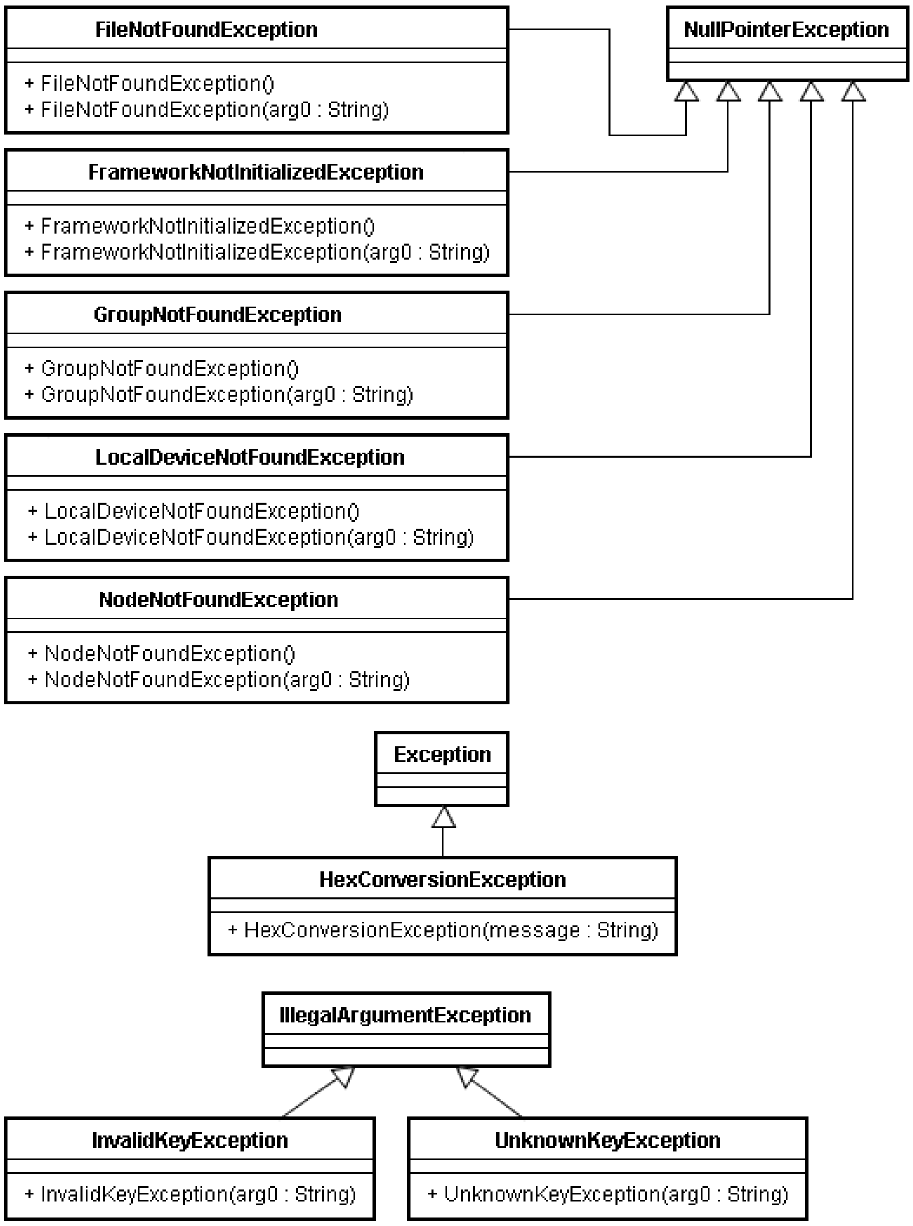

FileNotFoundException
+ FileNotFoundException()
+ FileNotFoundException(arg0 : String)
NullPointerException
FrameworkNotInitializedException
+ FrameworkNotInitializedException()
+ FrameworkNotInitializedException(arg0 : String)
GroupNotFoundException
+ GroupNotFoundException()
+ GroupNotFoundException(arg0 : String)
LocalDeviceNotFoundException
+ LocalDeviceNotFoundException()
+ LocalDeviceNotFoundException(arg0 : String)
NodeNotFoundException
+ NodeNotFoundException()
+ NodeNotFoundException(arg0 : String)
Exception
HexConversionException
+ HexConversionException(message : String)
IllegalArgumentException
InvalidKeyException
+ InvalidKeyException(arg0 : String)
UnknownKeyException
+ UnknownKeyException(arg0 : String)

Figure 16.30: *Class Diagram of the Exception Package*

16.14 The Log package

This package contains the Log functionality found in Peer2Me. The Log stores runtime-messages created at different locations in the framework. The log is useful for debugging as well as getting real-time information at runtime of the frameworks progress and well-being. The log package's contents are structured in layers as shown in Figure 16.31.

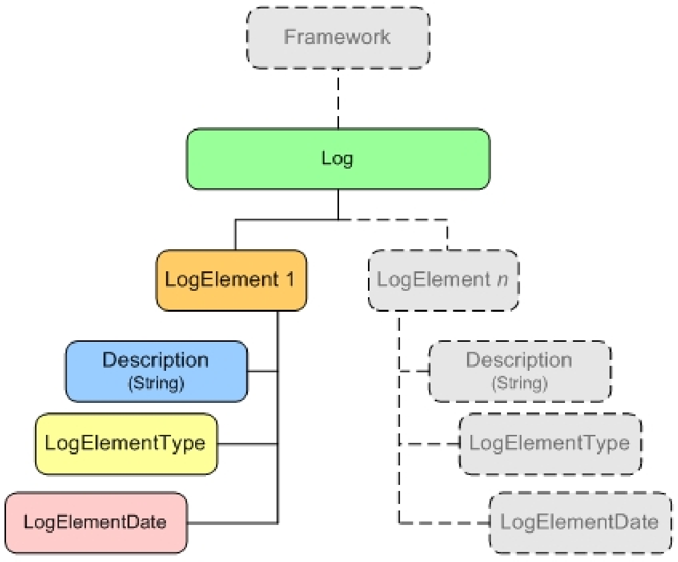


Figure 16.31: *The structure of the log system*

A log-entry is divided into four categories: *Error*, *Warning*, *Information* and *MIDlet*, defined by *LogElementType*. The three first categories described logentries arisen within the framework, the last one is intended to be used by the application running the framework.
Each entry will also carry information about time and date. These can be switched off as described in Section 16.14.1. The actual logging can also be switched off if the device running the framework has limited memory resources.
Four log-classes defines the log system, and is described beginning at Section 16.14.1.

16.14.1 Class: Log

The *Log* class is the base of the Log system. This class defines a *Vector* storing *LogElements* (described in Section 16.14.2), and provides methods for adding and retrieving entries, or clearing the Log. The class also provides an enumeration-capability for iterating through the entries. These methods are created with the aid of an internal counter keeping the track of entry last displayed. Using the methods `getNextElement()` or `getPreviousElement()` will return *null* if there are no more entries respectively first or last. Methods like `getFirstElement()` or `getLastElement()` can also be used.
If only a specific type of entry is wanted, `getElements(LogElementType type)` can be used, returning an array of LogElements of the specified type.

As mentioned, Log can be reached from all over the framework and the application due to its *Singleton* pattern. Adding a new entry in the log is therefore quite simple. Getting a reference to the Log-object is done by calling the `getInstance()`-method, and thereby calling the `addElement(LogElementType type, String description)`. This is shown in Listing 16.13.

Listing 16.13: *Example of adding an entry to the log*

```
//Example #1 - Adding an information entry
String description = "The framework is initialized";
Log.getInstance().addElement(LogElementType.INFORMATION,description);

```

```
//Example #2 - Adding a warning
Log myLog = Log.getInstance();
String description = "Could not get local name. Default set.";
myLog.addElement(LogElementType.WARNING,description);
```

After adding an entry to the log, it will be stored ready to be retrieved by using the get-methods previously mentioned which returns a *LogElement*, or the methods `getElementsAsList(..)` which returns a *javax.microedition.lcdui.List* object ready to be used in the graphical interface of a MIDlet. Input to these methods are title and/or which types of entries wanted in the list. These methods can prove to be quite useful, since it will reduce the amount of code needed in the MIDlet. We used these methods frequently during development for debugging purposes.
Listing 16.14 shows an example of how entries can be retrieved from the log.

Listing 16.14: *Example of retrieving entries from the log*

```
//Example #1 - Getting all elements
Log myLog = Log.getInstance();
LogElement[] allElements = myLog.getElements();

//Example #2 - Getting only warnings
Log myLog = Log.getInstance();
LogElement[] warningElements = myLog.getElements(LogElementType.WARNING);

//Example #3 - A List to show in GUI
display = Display.getDisplay(this);
Log myLog = Log.getInstance();
List list = myLog.getElementsAsList("MyLog");
display.setCurrent(list);
```

As these methods retrieve entries from the log created both by the framework and in the application using the framework, one must keep in mind to be careful when adding entries in the application using other *LogElementTypes* than *MIDlet*, since this might be confusing when mixed with the framework's native logentries.

Altering the properties *showDate* and *showTime* with their respective exposed methods will change the default toString of a LogElement. This is described in Section 16.14.2.

16.14.2 Class: LogElement

A *LogElement* holds information about the actual log-entry, and its constructor is not visible to application developer, since it is only created by *Log*. When retrieving a LogElement and displaying its toString() property, it contains the *date*, *time*, *type* and *description*. If desired, this can be customized by the methods `showDate(boolean show` and `showTime(boolean show)`.

16.14.3 Class: LogElementDate

This class holds only a time description for the actual *LogElement* it is created by. It uses the *java.util.Date* to get a date/time representation, and `split(String text,char splitcharacter)` in the class *TextUtil* to get date and time on the correct format. This could be achieved by using *java.util.Calendar* object, but this will result in unnecessary memory usage, and reduce the performance since *Calendar* is a complex object.

16.14.4 Class: LogElementType

A single log entry is defined as a type, a *LogElementType*, and as mentioned above it describes the *severity* or *category* of the entry. There are four available types:

- **Information:** Useful information generated mainly in the framework
- **Warning:** Typical warnings generated by the framework. Severity medium.
- **Error:** Critical errors or exceptions caught within the framework
- **MIDlet:** Intended for the MIDlet using the framework

They are all constructed to imitate *enums* in Java 5, enabling the use of *objects* as switches instead of *integers*. This not a big thing if the types are only used to distinct the types from each other when programming, but when it comes to displaying them, there is a difference. While using the typical approach, integers as identifiers, displaying the chosen value would output i.e. a number which does not say much if you have not got a translation table. Our approach to this is creating *protected classes* inheriting *LogElementType* and overriding the `toString()`-method. Then when calling i.e. `LogElementType.WARNING`, it instantiates a new object of the type *Warning*, storing this in *LogElement*. Displaying this would give the output "Warning".

16.14.5 Class Diagram

This section show the class diagram of the log package in Figure 16.32.

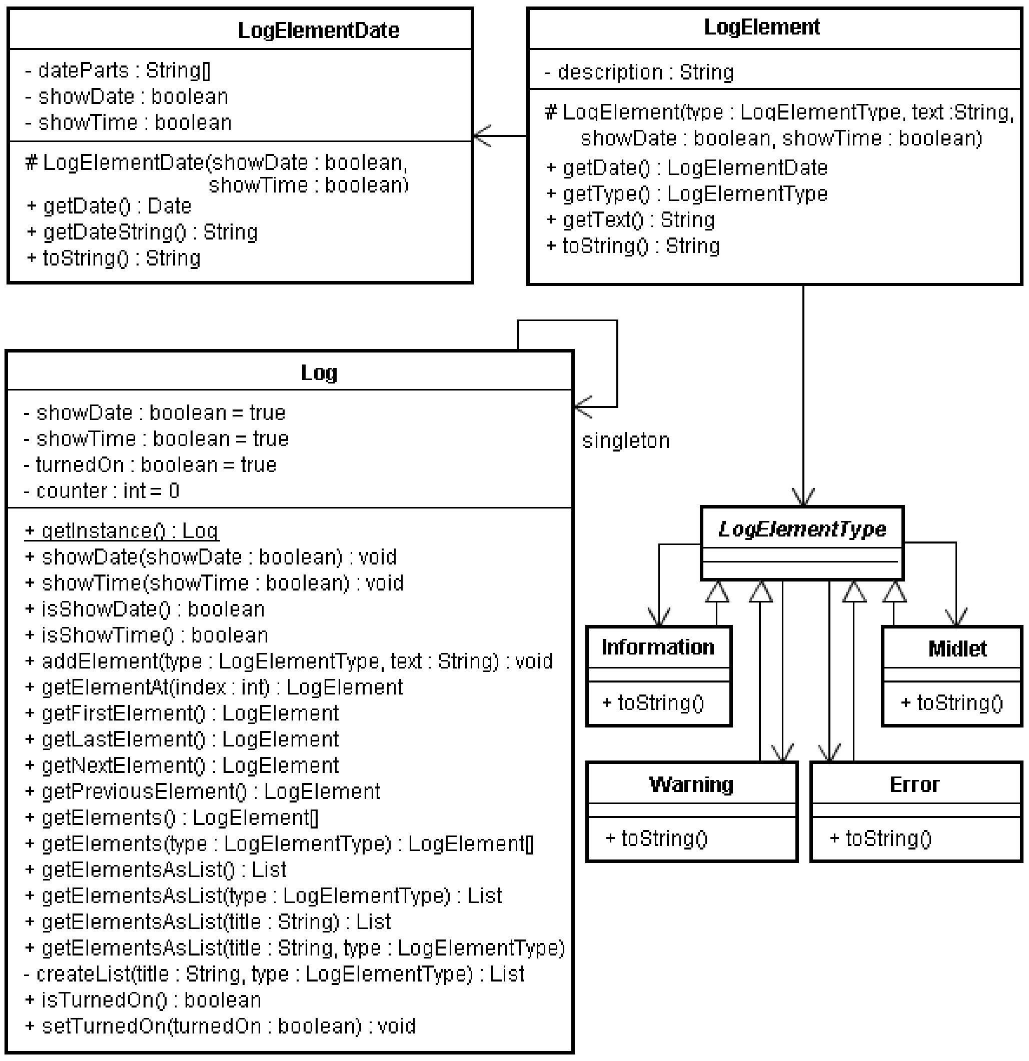

LogElementDate
- dateParts : String[]
- showDate : boolean
- showTime : boolean
# LogElementDate(showDate : boolean, showTime : boolean)
+ getDate() : Date
+ getDateString() : String
+ toString() : String
LogElement
- description : String
# LogElement(type : LogElementType, text :String, showDate : boolean, showTime : boolean)
+ getDate() : LogElementDate
+ getType() : LogElementType
+ getText() : String
+ toString() : String
Log
- showDate : boolean = true
- showTime : boolean = true
- turnedOn : boolean = true
- counter : int = 0
+ getInstance() : Log
+ showDate(showDate : boolean) : void
+ showTime(showTime : boolean) : void
+ isShowDate() : boolean
+ isShowTime() : boolean
+ addElement(type : LogElementType, text : String) : void
+ getElementAt(index : int) : LogElement
+ getFirstElement() : LogElement
+ getLastElement() : LogElement
+ getNextElement() : LogElement
+ getPreviousElement() : LogElement
+ getElements() : LogElement[]
+ getElements(type : LogElementType) : LogElement[]
+ getElementsAsList() : List
+ getElementsAsList(type : LogElementType) : List
+ getElementsAsList(title : String) : List
+ getElementsAsList(title : String, type : LogElementType)
- createList(title : String, type : LogElementType) : List
+ isTurnedOn() : boolean
+ setTurnedOn(turnedOn : boolean) : void
singleton
LogElementType
Information
+ toString()
Midlet
+ toString()
Warning
+ toString()
Error
+ toString()

Figure 16.32: *Class Diagram of the Log Package*

16.15 The Util package

The Util package acts as a stand-alone package with two classes needed to perform specific and rare tasks. These can be reached from anywhere within the framework with a direct reference, i.e. the class *Textutil* and the static method `split(String text, char splitcharacter` is used in *LogElementDate* directly. This is done despite our strict layered structure pattern, but in this case it is necessary with direct references since centralizing the utilities would create overhead and unwanted complexity.

16.15.1 Class: TextUtil

At this moment, this class contains only one method, `static String[] split(String text, char splitcharacter` which mimics the *split* method provided in J2SE. The basic construction is dividing the inputstring into an array of chars, then locating the splitcharacter, returning an array of strings with the chunks found. The method will return a null-string-array if the splitcharacter is not found.

16.15.2 Class: FileHandler

FileHandler is created using *the Singleton pattern*, described in Section 12.6.1, and is used store information about the mobile phone's filereceive-folder. This is done by getting the phone's root directories using `getRootDirectories()`, and adding the given **prefix**. The prefix is a static String sat to **file:///**.
If the received file folder is not sat explicitly, which is recommended, the folder is sat to (and created if necessary), **file:///<root>/tempp2me**, where <root> is the first root directory found on the mobile phone.

16.15.3 Class Diagram

This section shows the class diagram of the util package in Figure 16.33.

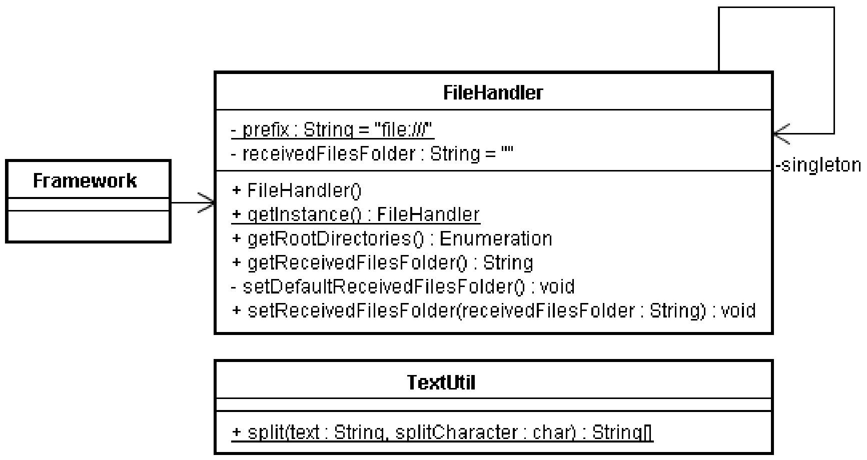

FileHandler
- prefix : String = "file:///"
- receivedFilesFolder : String = ""
+ FileHandler()
+ getInstance() : FileHandler
+ getRootDirectories() : Enumeration
+ getReceivedFilesFolder() : String
- setDefaultReceivedFilesFolder() : void
+ setReceivedFilesFolder(receivedFilesFolder : String) : void
-singleton
Framework
TextUtil
+ split(text : String, splitCharacter : char) : String[]

Figure 16.33: *Class Diagram of the Util Package*

Part V

Applications

CHAPTER 17

Introduction

This part describes three different applications that uses the new Peer2Me framework. The source code for the applications are listed in the Appendices.

The three applications:

1. **Chat2Me -** This is a kind of ”multichat” application. All mobile phones that runs this application will be able to chat with each other. If a participant writes a message, it will be sent to all the phones in the network. It follows the same principle as a classic ”group meeting” in MSN Messenger. The application consist of only one file containing 131 lines of code: *Chat2Me.java*.
2. **FilePusher -** This application lets you search for nearby devices running the same application and send a file to one of them. The application consists of two files: *FilePusher.java* and *FileBrowser.java*. The latter file contains the logic for browsing the filesystem on the mobile phone.
3. **ComplexMessageDemo -** This application illustrates how a message can contain a lot of different information and how the information can be retrieved on the receiving device. This application consist of one file: *ComplexMessageDemo*.

CHAPTER 18

The Applications

This chapter presents three applications that can be used to get familiar with the framework and is recommended reading for anyone who wants to create MIDlets using the new Peer2Me framework.

18.1 Chat2Me

Chat2Me demonstrates how easy a multichat MIDlet can be created using the Peer2Me framework. The MIDlet uses some basic J2ME functionalities such as displaying forms, letting the user input some text in a box and implements a CommandListener so that special functionalities is applied to the menubuttons. Only a small portion of code make up the necessary framework interaction which is explained in Section 18.1.1.

18.1.1 Walkthrough

Firstly, the class needs to import some classes from the framework, see Listing 18.1.

Listing 18.1: *Necessary import declarations*

```
import peer2me.framework.Framework;
import peer2me.framework.FrameworkSubscriber;
import peer2me.message.Message;
import peer2me.network.bluetooth.Bluetooth;
import peer2me.node.Node;
```

The class must implement the *FrameworkSubscriber* and have a *Framework* variable, see Listing 18.2.

Listing 18.2: *FrameworkSubscriber and Framework instance*

```
public class Chat2Me extends MIDlet implements FrameworkSubscriber,
            CommandListener
{
 private Framework framework;
```

Then the framework needs to be initialized as shown in Listing 18.3.

Listing 18.3: *Initialize framework*

```
  framework = Framework.getInstance("MyGroup","Chat2Me",new Bluetooth(),this);
  framework.initialize();
```

After the above operations are done, the application can now be discovered by other phones running the same application. In order to find these, a search can be done as shown in Listing 18.4.

Listing 18.4: *Search*

```
    framework.search();
```

After the search has completed, the method `searchCompleted()` is invoked and the display on phone shows the main chat window where all messages appear as participants write them. If the user wants to write a message, he clicks the ”write” button, writes his message, and presses ”send”. The message will appear in the main window on all participants, including himself. Listing 18.5 shows how the application creates a *Message* and sends it.

Listing 18.5: *Send a message*

```
    Message message = new Message();
    message.addElement("message",chatMessage.getString());
    message.addElement("nick",nick);
    message.addRecipients(framework.getAllNodes());
    framework.sendMessage(message);

    dialog.append(nick + ": " + chatMessage.getString(),null);
    display.setCurrent(dialog);
    dialog.setSelectedIndex(dialog.size()-1,true);
```

When one application sends a message, all the other phones will receive the message and the method `messageReceived(Message message)` is invoked. Listing 18.6 shows how the application handles the incoming message and displays the contents in the main chat window.

Listing 18.6: *Receive a message*

```
  String from = message.getString("nick");
  dialog.append(from + ":" + text,null);
  dialog.setSelectedIndex(dialog.size()-1,true);
 }
```

The application contains other methods that must be implemented by the *FrameworkSubscriber* interface:

- nodeDiscovered(Node node) - Is invoked when a new node is found in the network
- nodeLost(Node node) - Is invoked when a node leaves the network
- messagePartReceived(int partNumber, int totalParts) - Is invoked each time a chunk of a *Message* is received. This method serves no purpose in this simple application since no messages needs to be split up in different chunks. In the FilePusher application, this method comes in handy.

That is all you need for creating a simple multichat application. The chat does not depend on any device and participants may join and leave as they please. No new search is ever necessary for the devices that are already in the network. This application uses one class, *Chat2Me.java*. The complete source code can be found in Appendix A.

18.2 File2Push

File2Push is a MIDlet that lets the user select a file and send it to another peer. The peer will receive the file in several chunks, depending on the size of the file. The peer will shown details about the file, when the entire file is received.

18.2.1 Walkthrough

File2Push instanciates and performs a search in the same way as the Chat2Me application, see Section 18.1. After a search has completed, the user can select ”Show Nodes” from the menu on the phone. When he or she selects a node, a custom filebrowser will appear, where the user can browse through all the files on the mobile phone. The user then has to select a file, which will invoke the `sendFile(String fullPath)` method in the MIDlet. The MIDlet then creates a message containing the file and sends it as shown in Listing 18.7.

Listing 18.7: *Create a message and attache a file*

```
public void sendFile(String fullPath)
{
 Message message = new Message();
 message.addFile("file1", fullPath);
 message.addElement("file",true);
 message.addRecipient(currentlySelectedNode);
 mainForm.append("Sending\n");
 framework.sendMessage(message);
}
```

The peer that receives the file, must know where to store the file, which is specified as shown in Listing 18.8.

Listing 18.8: *Set received files folder*

```
 framework.setReceivedFilesFolder("root1/temp/");
```

The listing above shows how to specify the folder on an emulator. If the MIDlet is running on a mobile phone a different path must be used. For the Sony Ericsson phones we have used during testing, we have specified this folder: *e:/MSSEMC/Media files/audio/*. When the peer has received the entire message, the `messageReceived(Message message)` method is invoked. Listing 18.9 shows how to extract information about the file it just received.

Listing 18.9: *Receiving message containg file*

```
public void messageReceived(Message message)
{
 FileInfo receivedFile = message.getFile("file1");
 String filename = receivedFile.getFileName();
 String fullPath = receivedFile.getFullPathToFile();
 long sizeInBytes = receivedFile.getFileSize();
 mainForm.append("A file has been received\nName: " + filename +
   "\nFull path: " + fullPath + "\nSize (bytes): " + sizeInBytes);
}
```

If the file is large, the file will be sent in chunks as explained in Section 16.12.1. File2Push shows how to follow the progress of these chunks. Each time a chunk is received, the MIDlet calculates how much it has received and shows the user the progress in percentage of completetion as illustrated in Listing 18.10.

Listing 18.10: *Receiving chunks of message*

```
public void messagePartReceived(String messageID, int part, int total)
{
 int percent = (int)(((double)part/(double)total) * 100);
 mainForm.deleteAll();
 mainForm.append("Received: " + percent + " %");
}
```

This application uses two classes, *File2Push.java* and *LocalFileBrowser.java*. The complete source code can be found in Appendix B.

18.3 ComplexMessageDemo

ComplexMessageDemo is a MIDlet that shows the possibilities of the message functionality that recides in the framework.

18.3.1 Walkthrough

ComplexMessageDemo instanciates and performs a search in the same way as the Chat2Me application, see Section 18.1. After a search has completed, the user can select ”Show Nodes” from the menu on the phone. When he or she selects a node, a predefined complex message is created and sent as shown in Listing 18.11. This specific application has no real practical value, but is mearly meant as an illustration of how a message can contain multiple elements.

Listing 18.11: *Creating a complex message*

```
public void sendComplexMessage()
{
 Message message = new Message();
 message.addFile("file1", file1);
 message.addFile("file2", file2);
 message.addElement("summertime",true);
 message.addElement("country","Norway");
 message.addElement("pi",3.14);
 message.addElement("age",25);
 message.addElement("achar",'x');
 short value = 2;
 message.addElement("ashort",value);
 long largenumber = 922337000;
 message.addElement("longnumber",largenumber);
 message.addSerializedObject("info",makeStuffObject());
 message.addRecipients(framework.getAllNodes());
 framework.sendMessage(message);
 mainForm.append("Sending\n");
}
```

As you can see, the application attaches two files that have been specified in the code itself. In the *File2Push* application, the user selected the file that was attached to the message. Also, a serialized object is attached to the message. Listing 18.12 shows how the message is received.

Listing 18.12: *Receiving a complex message*

```
public void messageReceived(Message message)
{
 String outPut ="";
 boolean containFiles = message.getBoolean("file");
 if(containFiles)
 {
  FileInfo receivedFile1 = message.getFile("file1");
  String filename1 = receivedFile1.getFileName();
  String fullPath1 = receivedFile1.getFullPathToFile();
  long sizeInBytes1 = receivedFile1.getFileSize();

  FileInfo receivedFile2 = message.getFile("file2");
  String filename2 = receivedFile2.getFileName();
  String fullPath2 = receivedFile2.getFullPathToFile();
  long sizeInBytes2 = receivedFile2.getFileSize();

  String country = message.getString("country");
  double pi = message.getDouble("pi");
  int age = message.getInt("age");
  char charValue = message.getChar("achar");
  short shortValue = message.getShort("ashort");
  long longValue = message.getLong("longnumber");
  MyObject myObject = (MyObject)message.getSerializedObject("info");
  MyObject myObject2 = myObject.getMyObject();
```

```

   outPut += "\nFile1\nName: " + filename1 +
   "\nFull path: " + fullPath1 +
   "\nSize (bytes): " + sizeInBytes1 +
   "\nFile2\nName: " + filename2 +
   "\nFull path: " + fullPath2 +
   "\nSize (bytes): " + sizeInBytes2 +
   "\nCountry: " + country + "\npi: " + pi +
   "\nage: " + age +
   "\ncharValue: " + charValue +
   "\nshortValue: " + shortValue +
   "\nlongValue: " + longValue +
   "\MyObject 1:\nName: " + myObject.getName() +
   "\nIntValue: " + myObject.getIntValue() +
   "\ndoubleValue: " + myObject.doubleValue +
   "\MyObject 2:\nName: " + myObject2.getName() +
   "\nIntValue: " + myObject2.getIntValue() +
   "\ndoubleValue: " + myObject2.doubleValue;
   mainForm.append(outPut);
  }
 }
```

This application uses two classes, *ComplexMessageDemo.java* and *MyObject.java*. The complete source code can be found in Appendix C.

Part VI

Evaluation

CHAPTER 19

Testing

The framework has been tested during development in order to get a satisfactory end result. The testing has taken place in the SUN Microsystems Wireless Toolkit Emulator, The Sony-Ericsson Development Kit and on mobile phones. The mobile phones used in testing have been a couple of K750i and a W800i. Even though a test runs perfectly with no problems on the emulators, it might not run at all on a mobile phone. It has been time-consuming detecting why it works on the emulator and not on the mobile phone. Nevertheless, we feel that we have created a good framework for developing peer-to-peer applications on mobile phones. Section 19.1 describes the results of the testing of the functional requirements set out in Section 14.1. Section 19.2 describes the results of the testing of the quality requirements set out in Section 14.2.

19.1 Functional Requirements Results

Some of the tests can be carried out simultanously, while some of the tests have to isolated. Since we are as mentioned in Section 5.2, using *evolutionary prototyping* as a guideline, the testing has been carried out *during* the implementation process. If the requirement was not fulfilled, we would do another loop and evaluate again.
The results are presented in Table 19.1, and shows the results of testing the Peer2Me framework together with a custom testapplication suited for testing all the requirements on at least three mobile phones simultaneously.

19.2 Quality Requirements Results

This section describes how well we have implemented the quality requirements we set out in Section 14.2. Each requirement is given a status result and a comment to elaborate on how the requirement has been fulfilled.

Requirement: M1 Implement a workable network component in Peer2Me
Covered: Yes
Comments: The main architectural pattern in the Peer2Me framework is based on a layered model. We have currently implemented Bluetooth as a network technology and it should be possible to implement other network technologies as WLAN when the API becomes available for J2ME.

Requirement	Description	Covered
FR 1	The framework must support mobile phones	Yes
FR 2	The framework must be able to connect to other nodes	Yes
FR 3	The framework must support creation of ad hoc networks	Yes
FR 4	The framework must be able to connect to an existing ad hoc network	Yes
FR 5	Nodes in the network must be able to exchange messages	Yes
FR 6	It must be possible to store primitive variables (String, int, double, long, etc) in a message	Yes
FR 7	It should be possible to store serialized java objects in a message	Yes
FR 8	It must be possible to attach files to a message	Yes
FR 9	It should be possible to store primitive variables, serialized objects and files in one message	Yes
FR 10	The framework must track the origin of a message in order to enable a messagereply	Yes
FR 11	The framework must be able to create an open group	Yes
FR 12	The framework must simulate a pure P2P network	Yes
FR 13	The framework must hide the underlying network technology	Yes
FR 14	The framework must offer an easy way to create a log that can be viewed in the mobile phone during runtime	Yes
FR 15	The framework should register when a nodes leaves the network	Yes

Table 19.1: *Test results of the functional requirements*

Requirement: U1 Inform the application developer of exceptions
Covered: Yes
Comments: Method and classes that might fail due to events out of control of the framework will throw custom exceptions that can be caught in application that utilizes the framework. We have not yet experienced that the framework has become unavailable due to exceptions or errors.

Requirement: U2 Aid the implementation of Peer2Me i applications
Covered: Yes
Comments: All classes, interfaces, methods and parameters have been given descriptive names to help support easy interpretation and implementation for the developers point of view. The entire framework has also been documented with the de facto standard of Java documentation, Javadoc [23]. In addition, this masterthesis contains in depth information for the keen developer who wants to know more about the framework.

Requirement: T1 Log file
Covered: Yes
Comments: This was probably the first quality requirement that was implemented. During development we used this logging feature extensively in order to debug our code. Without this, debugging would have been a nightmare on mobile phones.

Requirement: A1 Normal disconnection of node
Covered: Yes
Comments: A node can easily disconnect by simply invoking a method on the framework.

The framework then sends a message to all the other nodes that it now disconnects.

Requirement: A2 Abnormal disconnection of node
Covered: Yes
Comments: In order to fulfill this requirement, we had to implement some sort of availability tactic. We used the ping/echo tactic for this purpose. If a node is lost, the framework will detect the lost node within a time frame of 30 seconds.

Requirement: A3 A new node arrives in an established network
Covered: Yes
Comments: If a new node wants to join an existing network, it can simple invoke a search and will join the network automatically.

CHAPTER 20

Comparison of Old and New Peer2Me

This chapter compares the old versus the new framework. First some empirical data such as total number of codelines, class, packages etc will be compared. Then we compare the requirements for both frameworks. Lastly, we illustrate how the new framework is put to use, and show the reader that the new framework is far easier to use than the old framework.

20.1 Statistical Comparison

Table 20.1 compares the old and new version of Peer2Me.

Aspect:	**Old Peer2Me**	**New Peer2Me**
Lines of code:	1875	3605
Number of classes:	29	52
Number of interfaces:	6	2
Number of packages:	18	11
Maximum inheritance tree depth:	6	5
Size of deployed jar file:	47,2 KiloByte	71,2 KiloByte

Table 20.1: *Statistical data about the old and new version of the Peer2Me framework*

The new framework contains over three times as much more code as the old framework. This does not mean that the old framework contains more efficient code, but that the new framework contains more functionality, especially when it comes to messages. The new framework also has twice as many classes, but less packages. We think that the old framework has too many packages which make it overly complex to navigate through. A slight increase in available memory would be required by the new framework as opposed to the old one. This would not be noticeable on newer mobile phones.

20.2 Functional Requirements Comparison

The two versions have both similar and different functional requirements. Table 20.2 shows the requirements that only resides in the old framework. Table 20.3 shows the requirements that only resides in the new framework. Table 20.4 shows the requirements that resides in both frameworks

Requirement	**Covered**
The system must support the creation of closed groups	Yes
The framework must include a mechanism for storing and retrieving objects	Yes
The system must support to allow a node to try to join a closed group	Yes
The system must allow users in a closed group to reject other nodes to join the group	Yes
The system must be able to present decision messages to the user	Partially

Table 20.2: *Requirements that only exists in the old framework*

Requirement	**Covered**
It must be possible to store primitive variables (String, int, double, long, etc) in a Message	Yes
It should be possible to store serialized java objects in a message	Yes
It must be possible to attach files to a message	Yes
It should be possible to store primitive variables, serialized objects and files in one message	Yes
The framework must simulate a pure P2P network	Yes
The framework must offer an easy way to create a log that can be viewed in the mobile phone during runtime	Yes
The framework should register when a nodes leaves the network	Yes

Table 20.3: *Requirements that only exists in the new framework*

	Covered	
Requirement	**Old**	**New**
The framework must support mobile phones	Yes	Yes
The framework must be able to connect to other nodes	Yes	Yes
The framework must support creation of ad hoc networks	Yes	Yes
The framework must be able to connect to an existing ad hoc network.	Yes	Yes
Nodes in the network must be able to exchange messages	Yes	Yes
The framework must track the origin of a message in order to enable a messagereply	Yes	Yes
The framework must be able to create an open group	Yes	Yes
The framework must hide the underlying network technology	Yes	Yes

Table 20.4: *Requirements that exists in both framework*

The old Peer2Me framework had the possibility to create closed groups and store objects to the phone's recordstore. We did not create closed groups since we had other priorities such as creating an advanced message system, filehandling and a pure ad hoc P2P network. Storing data in the phone's recordstore is an easy task which we did in two applications in our depthstudy last fall, [5]. We chose not to include this requirement in the new framework, since the recordstore has nothing to do with a peer-to-peer network, which is what Peer2Me really is about. The recordstore should be used to store data that the application can retrieve in subsequent executions and we leave this up to the application developer.

20.3 Differences that the Application Developer Experience

A great improvement from the old to the new framework, is the way the developer utilizes the framework. Listing 20.1 shows how a MIDlet needs to implement the different interfaces in the old framework and how it needs to be initialized.

Listing 20.1: *How to implement the old framework*

```
public class PanIm extends MIDlet implements GroupMonitor ,
 GroupDiscoveryListener , ExceptionHandler , MessageSubscriber
 {
 Framework framework;
 Group groups;
 Service service;
 String role;

 protected void startApp() throws MIDletStateChangeException
 {
  /*Initial GUI code removed
  *First the user has to select role, either master or slave*/
  groups = new Hashtable();

  if(role.equals("Slave"))
   chatForm = new ChatForm("PAN IM",this,"Slave");

  if(role.equals("Master"))
   chatForm = new ChatForm("PAN IM",this,"Master");

  service = new Service(RECORD_STORE);

  framework = Framework.getInstance(nickName, firstName + " " + lastName
    , "no.ntnu.idi.mowahs.project.bluetooth.network.BluetoothNetwork");
  framework.init();
  framework.setGroupDiscoveryListener(this);
  framework.setMessageSubscriber(this);
  framework.setExceptionHandler(this);

  if(role.equals("Master"))
  {
   group = new Group();
   group.setMaster(framework.getLocalNode());
   group.setMonitor(this);
   service.setGroup(group);
   group.setClosed(true);
   group.setService(service);
   foundNodes = new Hashtable();
  }
  framework.registerService(service);
  //......
```

Listing 20.2 shows how this is done in the new framework.

Listing 20.2: *How to implement the new framework*

```
public class Chat2Me extends MIDlet implements FrameworkSubscriber, CommandListener
{
 private Framework framework;

 protected void startApp() throws MIDletStateChangeException
 {
  /*Removed GUI code*/
  framework = Framework.getInstance("MyGroup","Chat2Me",new Bluetooth(),this);
  framework.initialize();
  //......
```

As you can see, there is ridiculously much that needs to be done the get the framework up and running in the old framework. Also, the framework must be initialized differently depending on the role of the device, which must be either master or slave. The new framework does not require this from the user, it just starts.

Looking at amount of code needed in an application using the new framework as opposed to the old one, will reveal how much more easier it is to use. Taking an application such as *Chat2Me* and comparing it to one similar test application in the old framework, shows the difference. Table 20.5 shows the lines of code needed. They are also compared to a chatapplication, *BlueChat* developed by Ben Hui [18].

Application	**Total lines of code**	**Communication specific of codelines**
Chat2Me	138	16
PanIM	242	42

Table 20.5: *Comparing a chat application using the new Peer2Me framework to PanIm using the old framework and BlueChat from Ben Hui.*

The column *Total lines of code* shows the total number of codelines in the application, including codelines regarding the graphics and program logics, as well as codelines regarding the communication. The column *Communication specific lines of code* is the total number of codelines which deals with the communication. This can be calling upon methods in the framework. BlueChat uses a bit different approach since it does not import any framework, but uses several classes to handle the communication. The number given in the second column for BlueChat is therefore the number of codelines equivalent to Chat2Me.

Now over to the differences in sending messages. In both cases, a message contains some text that needs to be sent to the members of the network. Listing 20.3 shows how this is done in the old framework.

Listing 20.3: *How to send a message in the old framework*

```
Message message = new Message();
TextMessagePart text = new TextMessagePart();
text.setDescription("message");
text.setFieldValue("This is some text");
message.addMessageBodyPart(text);
message.addRecipientGroup(service.getGroup());
framework.sendMessage(message,service);
```

Listing 20.4 shows how this is done in the new framework.

Listing 20.4: *How to send a message in the new framework*

```
 Message message = new Message();
 message.addElement("message","This is some text");
 message.addRecipients(framework.getGroup("MyGroup"))
 framework.sendMessage(message);
```

We have tried to make the new messagingsystem as intuitive and as simple as possible, yet more flexible. See Section 16.12.1 for details about the new possibilities. Receiving messages shows the same type of differences as sending messages.

Both the old and the new framework require that the MIDlet implements certain interfaces. The old framework requires three interfaces, the new requires one. Table 20.6 shows methods that are must be implemented by both versions and that serve the same purpose.

The common purposes of the methods listed in Table 20.6 are:

Method in old Peer2Me	**Method in new Peer2Me**
nodeJoined(Group group, Node node)	nodeDiscovered(Node node)
nodeLeft(Group group, Node node)	nodeLost(Node node)
messageReceived(Message message)	messageReceived(Message message)

Table 20.6: *Similar methods in new and old Peer2Me*

- `nodeJoined` and `nodeDiscovered` - Is invoked by the framework when a new node appears in the network
- `nodeLeft` and `nodeLost` - Is invoked by the framework when a node leaves the network
- `messageReceived` - Is invoked by the framework when a message is received

Applications using the old framework must also implement these methods:

- `allowJoin(Group group, Node node)` - The application must accept a new node before it can join
- `groupDiscovered(Group group)` - A group has been discovered in the network
- `handleException(Exception e)` - An exception is thrown in the framework an is sent to this method
- `runningService(String serviceID)` - Checks to see if the given service is running on the local node

Applications using the new framework must also implement these methods:

- `searchCompleted()` - When a search has completed, this method is invoked
- `messagePartReceived(int partNumber, int totalParts)` - A message can be split into many parts due to size or complexity. This enables the application developer to create an informative progressbar when receiving messages

CHAPTER 21

Problems

This Chapter will outline some of the problems we encountered during this project. It will focus on describing how and why the problems did arise, and what we did to overcome them. This chapter may be essential to understand some of the hidden obstacles that can be uncovered when designing large software solution for mobile phones.

21.1 Emulators vs Mobile Phones

A set of *emulators* is distributed as a part of the Java Wireless Toolkit, and is meant for testing MIDlets and mobile software before deploying it to a mobile phone. These emulators are described in Section 6.2.

One might think that getting the MIDlet to run on an emulator is a fool-proof indication that the software will run smoothly on a mobile phone. This might be correct in some simple applications, but when it comes to software like a framework, there are great differences between a mobile phone and the emulator. The next sections will outline some of the main issues regarding using emulator vs. mobile phones.

21.1.1 Performance

The emulator running on a computer possesses the same computing power as the computer unless this is explicitly set in the toolkit's preferences, and there is no exact way of adjusting the performance of the emulator to meet the performance level of a mobile phone.

A good example is running several threads in the application. Running two or three threads, or even four thread may not offer any difficulties on the mobile phone, but exceeding this running i.e. ten threads may get the phone to stop responding or turning off. This will of course not happen using an emulator. It is therefore of great importance to reduce the number of simultaneous threads to a minimum.
Another example would be handling files. When reading a file using an emulator, the speed is high and latency low. Using a mobile phone, the speed is significant lower. Since there are many things that runs slower on a mobile phone, it can also be necessary to focus on synchronizing threads and operations.

During the implementation phase of this project we experienced the problems mentioned above, along with memory shortage. When debugging the software, we used the framework's Log functionality excessively causing a frequent *out-of-memory* exception. Difficulties like this will strongly depend on the mobile phone the application is running on.

21.1.2 Graphical User Interface

As often when creating software with an user interface, one must pay close attention to the layout of the components. Designing a user interface on the emulator will only give an indication on how it will look on a mobile phone. This will also depend on how the mobile phone handles the GUI-components. As a rule of thumb, the display on the phone will be smaller and can contain less elements than the emulator's display.

The lack of updating can also be noticeable if the application consumes much resources. As an example to this is bad responsibility when using a *javax.microedition.lcdui.Gauge* at the same time as doing intensive computing. This occurred frequently when we developed the framework.

21.1.3 Exception Handling and Debugging

One of the few advantages of using an emulator when developing mobile software, is that there is at all times a console window which can be written to by a simple `System.out.println(..)`. When running the application on a phone, there is no possibility to get real-time information without writing it to the display same way as when adding components. The same is for unhandled exceptions. When a unhandled exception arise from the software, you might experience that the mobile phone turns it self off, or the application exits. When using an emulator, the console prints out that there has been an unhandled exception. This is especially useful to uncover simple bugs that are hard to locate. Getting the *NullPointException* is a classic, but also more random occurring exceptions as *BluetoothStateException* can occur.

21.2 Understanding Bluetooth Limitations

This Section will underline important issues to keep in mind when developing application using Bluetooth. As we have experienced during this project, there are many poorly documented sides of Bluetooth. Bluetooth is a complex technology depending on many factors when using it from scratch, and can behave quite differently after small alterations in the software. Debugging software using Bluetooth is a time-consuming task, even with the aid of emulators.

21.2.1 Master vs. Slave

There are strict rules concerning the roles when communicating via Bluetooth. In any session between two mobile phones there must be a *master* and a *slave*. This is also mentioned in Section 11.3.1. When *User A* wants to send a message to *User B*, *A* becomes the master, and *B* the slave. Normally, this setup does not have to change if *B* wants to respond with a message. This can however generate unusual circumstance with more than two participants and multiple messagesending, and if user *B* does not need to respond at once. This will keep the connection busy. We discovered this during the second half of this project forcing an unwanted reimplementation of the actual communication-handling. We made the roles switch as the sender switched. In other words, if user *A* as a master was finished sending a message to *B*, and *B* wanted to responded, they would switch roles. This approach is quite different to the approach used in the previous version of Peer2Me, but is by far more stable and produces less overhead.

21.2.2 Number of Nodes

Bluetooth technology has several constraints as discussed in Section 11.3.1, and in addition to this, there is a restriction to the number of connection a Bluetooth unit can handle at the same time. A Bluetooth device can only have 7 *active* connections at the same time. This means that a Bluetooth device can know many other Bluetooth devices, but only communicate with seven of these simultaneous. The number can even be lower on some mobile phones. It is therefore important to uncover the limitations of the mobile device that should run the application, or even better, design the software flexible enough to handle such situations.

21.2.3 Interference

Due to the fact that Bluetooth is a widespread technology embedded in many devices, it is not uncommon to discover devices that you do not want communication with. During a device search with a subsequent service search you might discover device that will make your search throw an exception. This is because the device responds to your request in peculiar way. A typical example of this is when discovering a computer with a USB-dongle with high security restrictions. Even though an exception is thrown, the search will continue. There is no particular way to avoid this issue.

21.2.4 Security Issues

Since Bluetooth has its own software included in the mobile phone's software, unwanted actions can occur when connecting to a new and unknown device. This is because the security mechanisms on the mobile phone interferes trying to pair the two phones. This will normally happen only once and will no affect the applications. The only problem we had with this, was that the software running when these messages appeared, continued running and did not halt. This gave at first problems synchronizing actions as one can not know how long the user of the phone needs to confirm the messages. This can perhaps be avoided with a signed MIDlet, but it may depend on the mobile phone.

Setting up a connection using Bluetooth can either be authenticated and/or encrypted. In this version of Peer2Me, neither is implemented. This is to keep the framework simple and well functioning.

CHAPTER 22

Answers to Research Questions

In this chapter we answer the research questions from Section 4.1.

1. Will a redesign of the framework make it easier to adopt when developing mobile ad-hoc applications in a J2ME development environment?

   (a) Can a pure peer-to-peer network be implemented in the new framework?

   Yes. We have successfully implemented a pure peer-to-peer network topology in the framework. The old framework only constituted a hybrid peer-to-peer network topology.

   (b) Is it possible to implement transfer of binary data in the new framework and what consequences will this yield?

   Yes. It is possible to transfer all types of files and data. The consequence is a slightly larger framework which does not have significant impact on the performance.

   (c) Will developers use less time to learn and create applications with the new framework compared to the old one?

   We compared how the applications must be implemented in the two different versions, it is no doubt that developers will spend less time learning how to use the new framework. current app. The new framework requires 2 lines and depends on no roles.

   (d) Will a redesign of the framework reduce the number of codelines, memory usage and dependencies in applications that uses the new framework compared to the old one?

   The number of codelines is drastically reduced as explained in the last answer. Slightly more memory is used, since the new framework is larger in size. The new framework *has one more required dependency due to filehandling possibilities, namely the JSR82 75 package.*

2. Would implementation of extra functionality make the framework more flexible and attractive?

   (a) What sort of functionality would add most value to the framework?

   The evaluation of the old framework in our depthstudy, [5], revealed the following desired functionality:

   - *Ability to send files, binary data and primitive datatypes between nodes*
   - *A pure peer-to-peer network technology that distributes network load and increases the availability and stability of the network drastically*

   (b) Will the extra functionality make the new framework incompatible with mobile phones that the old framework supports?

   Yes. It is not possible to implement a filehandling functionality without introducing the JSR82 75 package. Older phones such as the SonyEricsson P900 supports the Bluetooth API (JSR82 82), but not JSR82 75. The old framework was compatible with this phone, which the new framework is not.

3. What sort of impact would new technology or updates in existing technology have on the Peer2Me framework?

   (a) Will the technology make mobile ad hoc collaboration more efficient in terms of discoverytime, range and transfer rates?

   As mentioned under "Bluetooth Updates" in Section 11.3.1, everything indicates that the new technology will make mobile ad hoc collaboration more efficient when the new chip arrives the market. The press release by the Bluetooth SIG does not mention discoverytime, but the specification may be available in the near future. If this new chip is integrated in mobile phones, the Peer2Me framework will surely become even more attractive.

CHAPTER 23

Summary

This chapter will summarize the master thesis focusing on achievements, and further work.

23.1 Conclusion

Our goal for this master thesis was to create a new and improved Peer2Me framework. After an evaluation of the old framework in our depthstudy [5], we found that a total redesign of the framework could be beneficial. When designing the new framework, we paid special attention to the problems that Lund and Norum had experienced in their master thesis [31] in addition to our evaluation last fall. Our most important goal was to solve the dependency of a master node in the network. The old framework was based on a hybrid peer-to-peer network topology. We wanted to implement support for pure peer-to-peer network, which we successfully accomplished. The task was not a simple one as explained in Chapter 21. The new framework is also much easier to start using and requires fewer codelines than the old framework. Lund and Norum pointed out the messaging system in their framework as the weakest point. We recognized this as well and have come up with a new way of creating, sending and receiving messages. The new message system also supports the reading and transfer of files and serialized java objects. This opens up new possibilities that old framework did not have.

The new framework has grown in size, but also in functionality, scalability and usability which is three important aspects to consider. It is not compatible with older phones, but we expect that by the time the framework is ready for a wide spread use, most phones should support the framework. The most significant set-back for creating ad hoc mobile networks with Bluetooth is the discoverytime [1] which can be anything from 7 seconds to 20 depending on the device. Ad hoc networks should be created instantaneously to attract end-users and developers. However, the Bluetooth SIG (Special Interest Group) is planning on creating a new ultrawide band chip together with WiMedia Alliance that will have higher performance than todays chips. The chip will not consume any more power than todays Bluetooth chips, so that it can be incorporated with mobile devices and maintain it's current market segments.

It is very important that potential students who will follow up this project, pay special

[1]Discoverytime: The time it takes for a device to perform a search for all Bluetooth devices in the vicinity.

attention to the development of Ad Hoc Networking API (JSR82 259) which is lead by Sun Microsystems [35]. The goal of that project is to define an API that enables communication between mobile devices in a peer-to-peer ad-hoc network environment. We are not sure if the API can be used to create scatternets since it is not finished yet. However, we are confident that the Peer2Me framework will cover areas that the JSR82 259 will not, such as group management, complex message handling and availability tactics used in framework.

23.2 Further Work

Although the Peer2Me framework in its new version works fine without any problems, it can still be refined. From our point-of-view, the framework is not ready for a full release, but will function as a base for beta-testing, since all the main functionality is implemented. This section will discuss what needs to be done to the framework, presented in short term goals and long term goals.

23.3 Short-term Goals

This section will describe the short term goals of the Peer2Me framework.

23.3.1 Optimizing Network

In our version of Peer2Me, only *OBEX*, described in Section 11.3.1, is implemented as the Bluetooth transportation protocol. This is a fool-proof method providing good flexibility and stability. This is somewhat on the cost of performance. While OBEX is a good method for sending some type of dataobjects, might *RFCOMM* have better performance when transporting large files and data structures. The best thing would be to implement both protocols, using the most appropriate at all times.

As a Bluetooth unit may have implemented several channels used for data transportation, using as many of these as possible would increase the availability and the total performance of the unit. The main disadvantage when using all available channels is if the mobile phone supports multitasking. The framework will then block the Bluetooth device for other applications, resulting in unwanted situations.

23.3.2 Configuration

To make the framework easier to optimize, it should be easier to configure. To have a central point of configuration would be desired as there are several variables throughout the framework which affects its behaviour. A simple solution to this would be to have a simple class, xml or text file with a list of parameters, which is read from in the framework. The only alteration would then just take place in the *configuration-file*. Of course a default configuration must be an alternative at all times.

23.3.3 Scatternet

Scatternet is described in Section 8.5, is typically present if more than three nodes is available in a network where two of them cannot connect directly to each other, only through a third node. Since this new version of Peer2Me is based on a pure peer-to-peer model, adding possibility for scatternets is not a difficult task. To achieve this, the nodes in the network need a forwarding mechanism. A mechanism that can forward messages and function as a relay. The Peer2Me framework can then act in accordance with Figure 8.5.

One big issue when creating a scatternet configuration, is the handling and delegation of responsibility. One must decide which node should relay messages if more than one is present and available. A best-practice in such scenarios would be to delegate by load. I.e. the node with the fewer tasks should be responsible for forwarding the message.
If implementing messageforwarding, our version of Peer2Me is fully compliant to scatternet.

23.3.4 Mobile Agents

A mobile agent is a composition of software and data which is able to migrate from one computing unit to another autonomously, and continue its execution on the destination unit. A prerequisite to this is a scatternet configuration. Mobile agents have some advantages over conventional agents as described in *Mobile Agents: Are They a Good Idea?* by Harrison et al. [17]. Five of these advantages are given here:

- Move computation to data, reducing network load.
- Asynchronous execution on multiple heterogeneous network hosts.
- Dynamic adaptation - actions are dependent on the state of the host environment.
- Tolerant to network faults - able to operate without an active connection between client and server.
- Flexible maintenance - to change an agent's actions, only the source (rather than the computation hosts) must be updated.

Mobile agents can be among others be applied in the following scenarios:

- Resource availability, discovery, monitoring.
- Information retrieval.
- Network management.
- Dynamic software deployment.

Peer2Me can be used for mobile agents if scatternet as in Section 23.3.3 is implemented. To do this, one must create serialized objects which contains a state and then send the object to a new device. On the destionation device, the object should be invoked and continue its tasks.

23.4 Long-term Goals

This section will describe the long term goals of the Peer2Me framework.

23.4.1 Adopt new technology

Currently only Bluetooth is supported as a network medkum for Peer2Me, but as more technologies are introduced on mobile phones, an effort should be made to support these in Peer2Me. As mentioned before, the network medium is modular and interchangeable. The current Bluetooth module should also be kept up to date as the Bluetooth specification evolves and is introduced in new versions on mobile phones.
It should also be possible to use two network mediums at the same time.

The JSR259 project by Java should also be monitored to see if Peer2Me can be made to comply with the specification and/or utilize the concepts in the JSR82.

23.4.2 Up-to-Date

A long term goal that is inevitable, is the need to keep the framework up-to-date. When the framework can benefit from at new Java version, new MIDP version etc., it is important that the framework is updated as soon as possible.

23.4.3 Empirical Work and Applications

To evaluate the value of mobile collaborative applications on mobile phones it should be conducted full scale end user testing. A full scale testing is only available with fully working and popular application that are wide spread around and adopted by a large population. This would be a sensible way to study how such applications affect the way humans collaborate using mobile phones.

There should also be a focus on designing, developing and testing new applications that seek to explore all sides of the framework.

Part VII

Appendix

APPENDIX A

Chat2Me Source Code

Listing A.1: *Chat2Me source code*

```
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.List;
import javax.microedition.lcdui.TextField;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

import peer2me.framework.Framework;
import peer2me.framework.FrameworkSubscriber;
import peer2me.message.Message;
import peer2me.network.bluetooth.Bluetooth;
import peer2me.node.Node;
import peer2me.log.Log;
import peer2me.log.LogElementType;

public class Chat2Me extends MIDlet implements FrameworkSubscriber,
           CommandListener
{
 private Framework framework;
 private Display display;
 private Form nickNameForm;
 private Form writeForm;
 private Command ok;
 private Command send;
 private Command search;
 private Command exit;
 private Command write;
 private Command log;
 private Command back;

 private TextField nickName;
 private TextField chatMessage;

 private String nick;
 private List dialog;

 protected void startApp() throws MIDletStateChangeException
 {
  display = Display.getDisplay(this);
  nickName = new TextField("Nickname","",20,TextField.ANY);
  exit = new Command("Exit",Command.EXIT,1);
  ok = new Command("Ok",Command.OK,1);
  send = new Command("Send",Command.OK,1);
  search = new Command("Search", Command.ITEM,1);
  log = new Command("Log", Command.ITEM,2);
  back = new Command("Back", Command.BACK,3);

  nickNameForm = new Form("Chat");
  nickNameForm.append("Enter nick:");
  nickNameForm.append(nickName);
  nickNameForm.setCommandListener(this);
  nickNameForm.addCommand(ok);
  nickNameForm.addCommand(exit);

  chatMessage = new TextField("Message","",100,TextField.ANY);
  writeForm = new Form("Chat");
  writeForm.append(chatMessage);
  writeForm.addCommand(send);
  writeForm.setCommandListener(this);

  write = new Command("Write",Command.OK,0);
  dialog = new List("Chat",List.IMPLICIT);
  dialog.addCommand(write);
  dialog.addCommand(search);
  dialog.addCommand(exit);
  dialog.setCommandListener(this);

  framework = Framework.getInstance("MyGroup","Chat2Me",new Bluetooth(),this);
  framework.initialize();
  dialog.addCommand(log);
  display.setCurrent(nickNameForm);
 }

 protected void pauseApp(){}

 protected void destroyApp(boolean arg0) throws MIDletStateChangeException
 {
  framework.clean();
 }

 public void commandAction(Command cmd, Displayable arg1)
 {
  if(cmd == ok)
  {
   nick = nickName.getString();
   display.setCurrent(dialog);
  }
  else if(cmd == send)
  {
   Message message = new Message();
   message.addElement("message",chatMessage.getString());
   message.addElement("nick",nick);
   message.addRecipients(framework.getAllNodes());
   framework.sendMessage(message);

   dialog.append(nick + ": " + chatMessage.getString(),null);
   display.setCurrent(dialog);
   dialog.setSelectedIndex(dialog.size()-1,true);
  }
  else if(cmd == search)
  {
   framework.search();
   dialog.removeCommand(search);
  }
  else if(cmd == write)
  {
   display.setCurrent(writeForm);
  }
  else if(cmd == log)
  {
   List loglist = framework.getLog().getElementsAsList();
   loglist.addCommand(back);
   loglist.setCommandListener(this);
   display.setCurrent(loglist);
  }
  else if(cmd == back)
  {
   display.setCurrent(dialog);
  }
  else if(cmd == exit)
  {
   framework.clean();
   this.notifyDestroyed();
  }
 }

 public void searchCompleted()
 {
  display.setCurrent(dialog);
 }

 public void messageReceived(Message message)
 {
  String text = message.getString("message");
  String from = message.getString("nick");
  dialog.append(from + ":" + text,null);
  dialog.setSelectedIndex(dialog.size()-1,true);
 }

 public void nodeDiscovered(Node node)
 {
  dialog.append(node.getNodename() + " has joined",null);
  try{ dialog.removeCommand(search); }
  catch (Exception e){}
 }

 public void nodeLost(Node node)
 {
  dialog.append("+ " + node.getNodename() + " has left +",null);
 }

 public void messagePartReceived(int partNumber, int totalParts){}
}
```

APPENDIX B

File2Push Source Code

Listing B.1: *File2Push.java*

```java
import java.util.Vector;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.List;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

import peer2me.framework.Framework;
import peer2me.framework.FrameworkSubscriber;
import peer2me.log.Log;
import peer2me.message.FileInfo;
import peer2me.message.Message;
import peer2me.network.bluetooth.Bluetooth;
import peer2me.node.Node;

public class File2Push extends MIDlet implements FrameworkSubscriber, CommandListener
{
 private Framework framework;

 private Display display;
 private Displayable parent;

 private List nodeList;

 private Form mainForm;

 private Vector nodesFound;

 private Command search;
 private Command showNodes;
 private Command sendFile;
 private Command log;
 private Command exit;
 private Command back;

 private boolean addShowNodes = false;
 private Node currentlySelectedNode = null;
```

```java

protected void startApp() throws MIDletStateChangeException
{
 Log.getInstance().showDate(false);
 Log.getInstance().showTime(false);
 display = Display.getDisplay(this);

 mainForm = new Form("Peer2Me Test");

 parent = mainForm;

 framework = Framework.getInstance("MyGroup","File2Push",new Bluetooth(),false,this);
 nodeList = new List("Nodes",List.IMPLICIT);
 nodesFound = new Vector();

 search = new Command("Search",Command.OK,1);
 showNodes = new Command("Show Nodes",Command.OK,1);

 sendFile = new Command("Send file",Command.OK,1);

 log = new Command("Log",Command.OK,1);
 exit = new Command("Exit",Command.EXIT,1);
 back = new Command("Back",Command.BACK,3);

 mainForm.setCommandListener(this);

 mainForm.addCommand(search);
 mainForm.addCommand(back);
 mainForm.addCommand(log);
 mainForm.addCommand(exit);
 display.setCurrent(mainForm);
 framework.setframeworkSubscriber(this);
 framework.initialize();
 framework.setReceivedFilesFolder("root1/temp/");
}

protected void pauseApp()
{}

protected void destroyApp(boolean arg0) throws MIDletStateChangeException
{}

private void search()
{
 framework.search();
}

private List showNodes()
{
 nodeList.deleteAll();
 Node[] nodes = framework.getAllNodes();
 for (int i = 0; i < nodes.length; i++)
 {
  if(nodes[i].getAddress() != framework.getLocalNode().getAddress())
   nodeList.append(nodes[i].toString(),null);
 }
 nodeList.setCommandListener(this);
 nodeList.addCommand(back);
 nodeList.addCommand(sendFile);
 return nodeList;
}

public void nodeDiscovered(Node node)
{
 nodesFound.addElement(node);
 mainForm.append("Found : "+node.getNodename() + "\n");
 if(addShowNodes == false)
 {
  mainForm.addCommand(showNodes);
```

```
   addShowNodes = true;
  }
 }

 public void nodeLost(Node node)
 {
  nodesFound.removeElement(node);
  mainForm.append("Lost node "+node.getNodename()+ "at "+node.getAddress());
 }

 public void commandAction(Command cmd, Displayable arg1)
 {
  if(cmd == search)
  {
   mainForm.append("Searching...\n");
   search();
  }
  else if(cmd == showNodes)
   display.setCurrent(showNodes());
  else if(cmd == log)
  {
   List list = framework.getLog().getElementsAsList();
   list.addCommand(back);
   list.setCommandListener(this);
   display.setCurrent(list);
  }
  else if(cmd == back)
   display.setCurrent(parent);
  else if(cmd == exit)
  {
   framework.clean();
   this.notifyDestroyed();
  }
  else if(cmd == sendFile)
  {
   currentlySelectedNode = (Node)nodesFound.elementAt(nodeList.getSelectedIndex());
   new LocalFileBrowser(this);
  }
 }

 public void sendFile(String fullPath)
 {
  Message message = new Message();
  message.addFile("file1", fullPath);
  message.addElement("file",true);
  message.addRecipient(currentlySelectedNode);
  mainForm.append("Sending\n");
  framework.sendMessage(message);
 }

 public void messageReceived(Message message)
 {
  FileInfo receivedFile = message.getFile("file1");
  String filename = receivedFile.getFileName();
  String fullPath = receivedFile.getFullPathToFile();
  long sizeInBytes = receivedFile.getFileSize();
  mainForm.append("A file has been received\nName: " + filename +
    "\nFull path: " + fullPath + "\nSize (bytes): " + sizeInBytes);
 }

 public void messagePartReceived(String messageID, int part, int total)
 {
  int percent = (int)(((double)part/(double)total) * 100);
  mainForm.deleteAll();
  mainForm.append("Received: " + percent + " %");
 }

 public Form getMainForm()
 {
  return mainForm;
 }

 public void searchCompleted()
 {
  mainForm.append("Search completed");
 }
}
```

Listing B.2: *LocalFileBrowser.java*

```
import java.util.*;
import java.io.*;
import javax.microedition.io.*;
import javax.microedition.io.file.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import peer2me.log.Log;
import peer2me.log.LogElementType;

/**
 * Demonstration MIDlet for File Connection API. This MIDlet implements simple
 * file browser for the filesystem avaliable to the J2ME applications.
 *
 */
public class LocalFileBrowser implements CommandListener
{

    private String currDirName;

    private Command view = new Command("View", Command.ITEM, 1);
    private Command back = new Command("Back", Command.BACK, 2);
    private Command exit = new Command("Exit", Command.EXIT, 3);

    private Image dirIcon, fileIcon;
    private Image[] iconList;
    private File2Push midlet;

    /* special string denotes upper directory */
    private final static String UP_DIRECTORY = "..";

    /* special string that denotes apper directory accessible by this browser.
     * this virtual directory contains all roots.
     */
    private final static String MEGA_ROOT = "/";

    /* separator string as defined by FC specification */
    private final static String SEP_STR = "/";

    /* separator character as defined by FC specification */
    private final static char   SEP = '/';

    public LocalFileBrowser(File2Push midlet)
    {
     this.midlet = midlet;
        currDirName = MEGA_ROOT;
        try {
            dirIcon = Image.createImage("/icons/dir.png");
        } catch (IOException e) {
            dirIcon = null;
        }
        try {
            fileIcon = Image.createImage("/icons/file.png");
        } catch (IOException e) {
            fileIcon = null;
        }
        iconList = new Image[] { fileIcon, dirIcon };
        try
        {
            showCurrDir();
        } catch (SecurityException e) {
            Alert alert = new Alert("Error",
                "You are not authorized to access the restricted API",
                null, AlertType.ERROR);
            alert.setTimeout(Alert.FOREVER);
            Form form = new Form("Cannot access FileConnection");
            form.append(new StringItem(null,
                "You cannot run this MIDlet with the current permissions. "
                + "Sign the MIDlet suite, or run it in a different security domain"));
            form.addCommand(exit);
            form.setCommandListener(this);
            Display.getDisplay(midlet).setCurrent(alert, form);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void commandAction(Command c, Displayable d) {
        if (c == view)
        {
            List curr = (List)d;
            final String currFile = curr.getString(curr.getSelectedIndex());
            new Thread(new Runnable() {
                public void run() {
                    if (currFile.endsWith(SEP_STR) || currFile.equals(UP_DIRECTORY)) {
                        traverseDirectory(currFile);
                    } else {
                     String fullPath = currDirName + currFile;
                        Log.getInstance().addElement(LogElementType.INFORMATION,
                          "Selected file: " + fullPath);
                        midlet.sendFile(fullPath);
                    }
                }
            }).start();
        }else if (c == back) {
            showCurrDir();
        } else if (c == exit)
        {
         Display.getDisplay(midlet).setCurrent(midlet.getMainForm());
        }
    }

    /**
     * Show file list in the current directory .
     */
    void showCurrDir() {
     final LocalFileBrowser fileBrowser = this;
     new Thread(new Runnable() {
      public void run() {
       Enumeration e;
       FileConnection currDir = null;
       List browser;
       try {
        if (MEGA_ROOT.equals(currDirName)) {
         e = FileSystemRegistry.listRoots();
         browser = new List(currDirName, List.IMPLICIT);
        } else {
         currDir = (FileConnection)Connector.open("file://localhost/" +
           currDirName);
         e = currDir.list();
         browser = new List(currDirName, List.IMPLICIT);
         // not root - draw UP_DIRECTORY
         browser.append(UP_DIRECTORY, dirIcon);
        }

        while (e.hasMoreElements()) {
         String fileName = (String)e.nextElement();
```

```
          if (fileName.charAt(fileName.length()-1) == SEP) {
           // This is directory
           browser.append(fileName, dirIcon);
          } else {
           // this is regular file
           browser.append(fileName, fileIcon);
          }
         }

         browser.setSelectCommand(view);
         browser.addCommand(exit);
         browser.setCommandListener(fileBrowser);

         if (currDir != null) {
          currDir.close();
         }
         Display.getDisplay(midlet).setCurrent(browser);
        } catch (IOException ioe) {
         ioe.printStackTrace();
        }
       }
         }).start();
     }

     void traverseDirectory(String fileName) {
     /* In case of directory just change the current directory
      * and show it
      */
         if (currDirName.equals(MEGA_ROOT)) {
             if (fileName.equals(UP_DIRECTORY)) {
                 // can not go up from MEGA_ROOT
                 return;
             }
             currDirName = fileName;
         } else if (fileName.equals(UP_DIRECTORY)) {
             // Go up one directory
             // TODO use setFileConnection when implemented
             int i = currDirName.lastIndexOf(SEP, currDirName.length()-2);
             if (i != -1) {
                 currDirName = currDirName.substring(0, i+1);
             } else {
                 currDirName = MEGA_ROOT;
             }
         } else {
             currDirName = currDirName + fileName;
         }
         showCurrDir();
     }
```

APPENDIX C

ComplexMessageDemo Source Code

Listing C.1: *ComplexMessageDemo source code*

```java
import java.util.Vector;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.List;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

import peer2me.framework.Framework;
import peer2me.framework.FrameworkSubscriber;
import peer2me.log.Log;
import peer2me.message.FileInfo;
import peer2me.message.Message;
import peer2me.network.bluetooth.Bluetooth;
import peer2me.node.Node;

public class ComplexMessageDemo extends MIDlet implements FrameworkSubscriber, CommandListener
{
 private Framework framework;

 private Display display;
 private Displayable parent;

 private List nodeList;

 private Form mainForm;

 private Vector nodesFound;

 private Command search;
 private Command showNodes;
 private Command send;
 private Command log;
 private Command exit;
 private Command back;

 private boolean addShowNodes = false;

private String file1 = "root1/images/christmas.jpg";
private String file2 = "root1/images/matrix.jpg";

protected void startApp() throws MIDletStateChangeException
{
 Log.getInstance().showDate(false);
 Log.getInstance().showTime(false);
 display = Display.getDisplay(this);

 mainForm = new Form("Peer2Me Test");

 parent = mainForm;

 framework = Framework.getInstance("MyGroup","Complex",new Bluetooth(),this);
 nodeList = new List("Nodes",List.IMPLICIT);
 nodesFound = new Vector();

 search = new Command("Search",Command.OK,1);
 showNodes = new Command("Show Nodes",Command.OK,1);

 log = new Command("Log",Command.OK,1);
 exit = new Command("Exit",Command.EXIT,1);
 back = new Command("Back",Command.BACK,3);
 send = new Command("Send",Command.OK,1);

 mainForm.setCommandListener(this);

 mainForm.addCommand(search);
 mainForm.addCommand(back);
 mainForm.addCommand(log);
 mainForm.addCommand(exit);
 display.setCurrent(mainForm);
 framework.setframeworkSubscriber(this);
 framework.initialize();
 framework.setReceivedFilesFolder("root1/temp/");
}

protected void pauseApp()
{}

protected void destroyApp(boolean arg0) throws MIDletStateChangeException
{}

private void search()
{
 framework.search();
}

private List showNodes()
{
 nodeList.deleteAll();
 Node[] nodes = framework.getAllNodes();
 for (int i = 0; i < nodes.length; i++)
 {
  if(nodes[i].getAddress() != framework.getLocalNode().getAddress())
   nodeList.append(nodes[i].toString(),null);
 }
 nodeList.setCommandListener(this);
 nodeList.addCommand(back);
 nodeList.addCommand(send);
 return nodeList;
}

public void nodeDiscovered(Node node)
{
 nodesFound.addElement(node);
 mainForm.append("Found : "+node.getNodename() + "\n");
 if(addShowNodes == false)
 {
   mainForm.addCommand(showNodes);
   addShowNodes = true;
  }
 }

 public void nodeLost(Node node)
 {
  nodesFound.removeElement(node);
  mainForm.append("Lost node "+node.getNodename()+ "at "+node.getAddress());
 }

 public void commandAction(Command cmd, Displayable arg1)
 {
  if(cmd == search)
  {
   mainForm.append("Searching...\n");
   search();
  }
  else if(cmd == showNodes)
   display.setCurrent(showNodes());
  else if(cmd == log)
  {
   List list = framework.getLog().getElementsAsList("Loggen");
   list.addCommand(back);
   list.setCommandListener(this);
   display.setCurrent(list);
  }
  else if(cmd == back)
   display.setCurrent(parent);
  else if(cmd == exit)
  {
   framework.clean();
   this.notifyDestroyed();
  }
  else if(cmd == send)
  {
   sendComplexMessage();
  }
 }

 public void sendComplexMessage()
 {
  Message message = new Message();
  message.addFile("file1", file1);
  message.addFile("file2", file2);
  message.addElement("summertime",true);
  message.addElement("country","Norway");
  message.addElement("pi",3.14);
  message.addElement("age",25);
  message.addElement("achar",'x');
  short value = 2;
  message.addElement("ashort",value);
  long largenumber = 922337000;
  message.addElement("longnumber",largenumber);
  message.addSerializedObject("info",makeStuffObject());
  message.addRecipients(framework.getAllNodes());
  framework.sendMessage(message);
  mainForm.append("Sending\n");
 }

 public void messageReceived(Message message)
 {
  String outPut ="";
  boolean containFiles = message.getBoolean("file");
  if(containFiles)
  {
   FileInfo receivedFile1 = message.getFile("file1");
   String filename1 = receivedFile1.getFileName();
   String fullPath1 = receivedFile1.getFullPathToFile();
```

```
    long sizeInBytes1 = receivedFile1.getFileSize();

    FileInfo receivedFile2 = message.getFile("file2");
    String filename2 = receivedFile2.getFileName();
    String fullPath2 = receivedFile2.getFullPathToFile();
    long sizeInBytes2 = receivedFile2.getFileSize();

    String country = message.getString("country");
    double pi = message.getDouble("pi");
    int age = message.getInt("age");
    char charValue = message.getChar("achar");
    short shortValue = message.getShort("ashort");
    long longValue = message.getLong("longnumber");
    MyObject myObject = (MyObject)message.getSerializedObject("info");
    MyObject myObject2 = myObject.getMyObject();

    outPut += "\nFile1\nName: " + filename1 +
    "\nFull path: " + fullPath1 +
    "\nSize (bytes): " + sizeInBytes1 +
    "\nFile2\nName: " + filename2 +
    "\nFull path: " + fullPath2 +
    "\nSize (bytes): " + sizeInBytes2 +
    "\nCountry: " + country + "\npi: " + pi +
    "\nage: " + age +
    "\ncharValue: " + charValue +
    "\nshortValue: " + shortValue +
    "\nlongValue: " + longValue +
    "\MyObject 1:\nName: " + myObject.getName() +
    "\nIntValue: " + myObject.getIntValue() +
    "\ndoubleValue: " + myObject.doubleValue +
    "\MyObject 2:\nName: " + myObject2.getName() +
    "\nIntValue: " + myObject2.getIntValue() +
    "\ndoubleValue: " + myObject2.doubleValue;
    mainForm.append(outPut);
   }
  }

  public void messagePartReceived(String messageID, int part, int total)
  {
   int percent = (int)(((double)part/(double)total) * 100);
   mainForm.deleteAll();
   mainForm.append("Received: " + percent + " %");
  }

  public void searchCompleted()
  {
   mainForm.append("Search completed");
  }

  private Information makeStuffObject()
  {
   Information s = new Information();
   s.setDoubleValue(7.89);
   s.setName("Kim");
   s.setIntValue(30);

   Information s2 = new Information();
   s2.setDoubleValue(14.9);
   s2.setName("Tommy");
   s2.setIntValue(25);
   s.setInformation(s2);
   return s;
  }
 }
```

APPENDIX D

Contents of Zip File

- **Applications**
  - Chat2Me
  - File2Push
- **Javadoc**
- **PDF**
  - This Masterthesis
  - Our Depthstudy
- **Peer2Me JAR file**
- **Source code**

Glossary

Best practice	A common way of doing things regarded as a good method.	74
Bluetooth	an industrial specification for wireless personal area networks (PANs)	3
GSM	Global System for Mobile Communications	3
IrDA	Infrared Data Association	3
J2ME	Java 2 Micro Edition	3
MANET	Mobile Ad Hoc Network	3
MIDP	Mobile Information Device Profile. Set of Java APIs that is generally implemented on the Connected Limited Device Configuration (CLDC)	43
MOWAHS	MObile Work Across Heterogeneous Systems	5
Multihop	Reaching outer nodes starting in center and jumping from node to node until destination is reached.	38
PDA	Personal digital assistant	3
Symbian OS	An operating system developed by Symbian found on many mobile phones in different versions.	39
WLAN	Wireless Local Area Network that uses radio waves as its carrier	3

Bibliography

[1] Sony Ericsson Mobile Communication AB. Java docs tools. Retrieved January 27th, 2006, from `http://developer.sonyericsson.com/site/global/docstools/java/p_java.jsp.`, 2006.

[2] Victor R. Basili. Experimental software engineering issues: Critical assessment and future directions. *The Experimental Paradigm*, 1992.

[3] Len Bass, Paul Clements, and Rick Kazman. *Software in Practice, Second Edition.* Addison Wesley, 2004.

[4] BEDD. Bedd^{TM}bringing people together. Retrieved November 2th, 2005, from `http://www.bedd.com/about.html.`, 2005.

[5] Tommy Bjørnsgård and Kim Petter Saxlund. Evaluation of peer2me. Technical report, Norwegian University of Science and Technology, 2005.

[6] Mitch Blaser. Industrial-strength security for zigbee: The case for public-key cryptography. 2005.

[7] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern Oriented Software Architecture.* Wiley, John Sons, Incorporated, 1996.

[8] Peter H. Carstensen and Kjeld Schmidt. Computer supported cooperative work: New challenges to systems of design. *Handbook of Human Factors*, 2002.

[9] Datacomm Research Company. Using mimo-ofdm technology to boost wireless lan performance today. Technical report, Datacomm Research Company, 2005.

[10] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed System: Concepts and Design.* Pearson: Addison Wesley, 2001.

[11] Sony Ericsson. Part one: Using irda with the obex protocol to create a client application 259: Ad hoc networking api. Retrieved April 24th, 2006, from `http://developer.sonyericsson.com/site/global/techsupport/tipstrickscode/java/p_part1_usingirda_obexprotocol.jsp.`, 2005.

[12] George H. Forman and John Zahorjan. The challenges of mobile computing. Technical report, University of Washington, 1994.

[13] The Eclipse Foundation. What is eclipse? Retrieved January 26th, 2005, from `http://www.eclipse.org/org/.`, 2005.

[14] Open Source Technology Group. Blue cove. Retrieved Desember 7th, 2005, from `http://sourceforge.net/projects/bluecove/.`, 2005.

[15] Open Source Technology Group. Eclipseme. Retrieved January 26th, 2005, from `http://sourceforge.net/projects/eclipseme/.`, 2005.

[16] Open Source Technology Group. Texlipse. Retrieved January 26th, 2005, from `http://sourceforge.net/projects/texlipse/`., 2005.

[17] Colin G. Harrison, David M. Chess, and Aaron Kershenbaum. *Mobile Agents: Are they a good idea?* IBM Research Division, 1995.

[18] Ben Hui. Connecting pc and phone with java bluetooth api part 1. Retrieved November 2th, 2005, from `http://www.benhui.net/modules.php?name=Bluetooth&page=Connect_PC_Phone_Part_1.html`., 2005.

[19] Team in a Box Ltd. Eclipse metrics plugin. Retrieved January 26th, 2005, from `http://www.teaminabox.co.uk/downloads/metrics/`., 2004.

[20] Cisco Systems Inc. Osi model. Website: `http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/introint.htm`% bibnodot., 2006.

[21] Sun Microsoystems Inc. Java 2 platform, micro edition (j2me) datasheet. Retrieved Mars 11th, 2006, from `http://java.sun.com/j2me/docs/j2me-ds.pdf`., 2002.

[22] Sun Microsoystems Inc. Java 2 platform, micro edition (j2me). Retrieved November 25th, 2005, from `http://java.sun.com/j2me/`., 2005.

[23] Sun Microsoystems Inc. Javadoc tool. Retrieved May 24th, 2006, from `http://java.sun.com/j2se/javadoc/`., 2006.

[24] Sun Microsystems Inc. Sun java webpage. Website: `http://java.sun.com/`., 2006.

[25] Lars Kirkhus and Anders R. Sveen. An examination of mobile devices for spontaneous collaboration. Technical report, The Norwegian University of Science and Technology (NTNU), 2003.

[26] Lars Kirkhus and Anders R. Sveen. Mowahs - mobile collaboration framework. Technical report, The Norwegian University of Science and Technology (NTNU), 2004.

[27] Gerd Kortuem, Jay Schneider, Dustin Preuitt, Thaddeus G. C. Thompson, Stephen Fickas, and Zary Segall. When peer-to-peer comes face-to-face: Collaborative peer-to-peer computing in mobile ad hoc networks. Technical report, Department of Computer and Information Science, University of Oregon, 2001.

[28] Adam Laurie and Ben Laurie. Serious flaws in bluetooth security lead to disclosure of personal data. Technical report, A.L. Digital Ltd, 2003.

[29] L.J.Bannon. The context of cscw. *Report of CoTech Working Group 4*, 1991-1992.

[30] Carl-Henrik Wolf Lund and Michael Sars Norum. A framework for mobile collaborative applications on mobile phones. Technical report, Norwegian University of Science and Technology, 2004.

[31] Carl-Henrik Wolf Lund and Michael Sars Norum. The peer2me framework, a framework for mobile collaboration on mobile phones. Master's thesis, NTNU, 2005.

[32] Qusay H. Mahmoud. Part ii: The java apis for bluetooth wireless technology. 2003.

[33] Nico Maibaum and Thomas Mundt. Jxta: A technology facilitating mobile peer-to-peer networks. Technical report, University of Rostock; Department of Computer Science, Germany, 2002.

[34] Sun Microsystems. Bluetooth sig selects wimedia alliance ultra-wideband technology for high speed bluetooth® applications. Retrieved April 24th, 2006, from `http://www.j2medev.com/api/btapi/javax/obex/HeaderSet.html`., 2004.

[35] Sun Microsystems. Jsr 259: Ad hoc networking api. Retrieved December 8th, 2005, from `http://www.jcp.org/en/jsr/detail?id=259`., 2005.

[36] Sun Microsystems. Sun java wireless toolkit. Retrieved January 26th, 2005, from `http://java.sun.com/products/sjwtoolkit/`., 2006.

[37] Ian Goldberg Nikita Borisov and David Wagner. Intercepting mobile communications: The insecurity of 802.11. Technical report, University of California, Berkeley, 2001.

[38] University of Tromsø. Open-obex. Retrieved February 24th, 2006, from `http://www.ravioli.pasta.cs.uit.no/open-obex.`, 2005.

[39] Christian Schenk. About miktex. Retrieved January 26th, 2005, from `http://www.miktex.org/about.html.`, 2006.

[40] SIG. The official bluetooth website. Retrieved November 26th, 2005, from `http://www.bluetooth.com/.`, 2005.

[41] Bluetooth SIG. Interface headerset. Retrieved May 24th, 2006, from `http://bluetooth.com/Bluetooth/Press/SIG/BLUETOOTH_SIG_SELECTS_WIMEDIA_ALLIANCE_ULTRAWIDEBAND_TECHNOLOGY_FOR_HIGH_SPEED_BLUETOOTH_APPLICATION.htm`bnodot., 2006.

[42] Wikipedia. Mobile ad-hoc network. Retrieved October 7th, 2005, from `http://en.wikipedia.org/wiki/Mobile_ad-hoc_network.`, 2005.

[43] Wikipedia. Open mobile alliance. Retrieved November 20th, 2005, from `http://en.wikipedia.org/wiki/Open_Mobile_Alliance.`, 2005.

[44] Wikipedia. Peer-to-peer. Retrieved September 25th, 2005, from `http://en.wikipedia.org/wiki/P2p.`, 2005.

[45] Wikipedia. Personal area network. Retrieved October 7th, 2005, from `http://en.wikipedia.org/wiki/Personal_area_network.`, 2005.

[46] Wikipedia. Syncml. Retrieved November 20th, 2005, from `http://en.wikipedia.org/wiki/SyncML.`, 2005.