NTNU
Innovation and Creativity

Redesign and optimalization of the Peer2Me Framework

A Framework for developing Applications supporting mobile collaboration using J2ME

Steinar A. Hestnes
Torbjørn Vatn

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

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

Problem Description

The main goal of the project is to redesign the Peer2Me framework originally developed by Sars Norum and Wolf Lund in their master thesis from spring 2005. Peer2Me is a framework for developing mobile collaborative applications on mobile phones utilizing Personal Area Networks (PANs). The framework is developed using J2ME technology and currently supports Bluetooth communication.

The Peer2Me framework has been tested and analyzed in two separate depth study projects, and several possible improvements have been discovered. This includes improving the architecture, simplifying the interface presented to the developer, and decrease its footprint.

To decide whether the improvements have been successful, the original and the redesigned Peer2Me framework will be thoroughly compared at the end of the project. The results of this comparison will be used to evaluate the quality of the redesigned framework.

Assignment given: 20. January 2006
Supervisor: Alf Inge Wang, IDI

Abstract

This project was started to develop a new improved version of the Peer2Me framework. After having evaluated the first version of the Peer2Me framework in our depth study project in the fall of 2005, quite a few possible improvements came up.

This report starts with an introduction to Computer Supported Cooperative Work (CSCW), wireless networking, Peer-to-Peer (P2P) computing, and mobile ad hoc networking. It also introduces some central concepts concerning design of a software architecture, and technology relevant to the development of the Peer2Me framework.

The redesign of the framework was started by eliciting a set of new requirements, constituting the basis for designing the new Peer2Me architecture. Through an iterative and incremental development process, Peer2Me framework v2.0 was developed with several new features. An instant messenger application has been developed using both versions of the framework, in order to compare them.

A thorough comparison of Peer2Me v1.0 and Peer2Me v2.0 shows that the redesign has resulted in a reduced framework footprint and complexity, a simplified interface towards the MIDlets, and a considerably increase in transfer rate.

Preface

This master thesis within Software Engineering was written by Torbjørn Vatn and Steinar A. Hestnes in the period from January 2006 to June 2006. It documents the work contributed to the Peer2Me project which is related to the MOWAHS (MObile Work Across Heterogeneous Systems) project run by the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU).

Acknowledgements

We would like to give credit to Alf Inge Wang for assigning us to this project and for his help and guidance. We would also like to thank Carl-Henrik Wolf Lund and Michael Sars Norum for their work on the Peer2Me framework through their depth study and master thesis.

Trondheim, 16.06.2006

Torbjørn Vatn

Steinar A. Hestnes

Contents

I Introduction **1**

1 Introduction **3**
1.1 Motivation . . . 4
1.2 Problem Definition . . . 4
1.3 Project context and limitation of scope . . . 5
1.4 Reader's Guide . . . 5
1.4.1 Which Chapters to Read . . . 7

2 Research Questions and Method **9**
2.1 Research Questions . . . 9
2.2 Research Method . . . 11
2.2.1 The engineering approach . . . 12
2.2.2 The empirical approach . . . 15
2.2.3 Evaluation . . . 17

II Prestudy **19**

3 Central Concepts **21**
3.1 Cooperative Work . . . 21
3.1.1 Computer Supported Cooperative Work . . . 22
3.1.2 Mobile Computer Supported Cooperative Work . . . 23
3.2 Wireless Networking . . . 24
3.2.1 Wireless Wide Area Networks (WWANs) . . . 24
3.2.2 Wireless Metropolitan Area Networks (WMANs) . . . 24
3.2.3 Wireless Local Area Networks (WLANs) . . . 24
3.2.4 Wireless Personal Area Networks (WPANs) . . . 24
3.3 Peer-to-Peer Computing . . . 25
3.3.1 Pure or hybrid peer-to-peer . . . 26
3.4 Mobile Ad Hoc Networking . . . 27

4 Software Architecture **29**
4.1 What is Software Architecture? . . . 29
4.2 Creating an Architecture . . . 30
4.3 Quality Attributes . . . 31

4.3.1 Usability . . . 31
4.3.2 Performance . . . 31
4.3.3 Modifiability . . . 31
4.3.4 Availability . . . 32
4.3.5 Security . . . 32
4.3.6 Testability . . . 32

5 Technology **33**
5.1 Java2 Micro Edition . . . 33
5.1.1 J2ME Architecture . . . 33
5.2 Bluetooth . . . 35
5.2.1 What is Bluetooth? . . . 36
5.2.2 Origin of the name . . . 36
5.2.3 Communicating via Radio Waves . . . 36
5.2.4 Bluetooth Transfer Rate . . . 37
5.2.5 Bluetooth Security . . . 38
5.2.6 Piconets and Scatternets . . . 38

6 The Original Peer2Me Framework **41**
6.1 Peer2Me v1.0 Domain Concepts . . . 41
6.2 Peer2Me v1.0 Functional Requirements . . . 42
6.3 Peer2Me v1.0 Non-functional Requirements . . . 42
6.4 Peer2Me v1.0 Design . . . 42
6.5 Known Problems . . . 44

7 Related Work **47**
7.1 JXTA . . . 47
7.1.1 JXTA-J2ME (JXME) . . . 48
7.1.2 Jadabs-CLDC . . . 48
7.2 Ergon - J2ME Wireless Application Framework . . . 48
7.3 BEDD . . . 49
7.4 JSR-259: Ad Hoc Networking API . . . 49
7.5 Conclusion . . . 50

III Redesigning the Peer2Me Framework v1.0 **51**

8 Requirements **53**
8.1 Functional Requirements . . . 53
8.1.1 Use Cases . . . 55
8.2 Non-functional Requirements . . . 64
8.2.1 Usability . . . 64
8.2.2 Performance . . . 65
8.2.3 Modifiability . . . 65
8.2.4 Availability . . . 65
8.2.5 Security . . . 66
8.2.6 Testability . . . 66
8.3 Environmental Requirements . . . 67

9 Design **69**
9.1 High Level Architecture . . . 69
9.2 Detailed description . . . 71
9.2.1 Framework package . . . 74
9.2.2 Domain package . . . 74
9.2.3 Network package . . . 75
9.2.4 Util package . . . 75
9.3 Design Patterns . . . 76

10 Implementation **77**
10.1 Implementation Method . . . 77
10.2 Implementation tools . . . 78
10.3 Source Code Examples . . . 80
10.3.1 The Framework interface . . . 81
10.3.2 The initFramework() method in FrameworkFrontEnd.java . . . 84
10.3.3 The getInstance() method in Network.java . . . 86
10.3.4 The addParticipant() method in Group.java . . . 88
10.3.5 The init() method in BluetoothNetwork.java . . . 89
10.3.6 The doDeviceDiscovery() method in BluetoothServiceDiscovery.java . . . 91
10.3.7 The notifyAboutFoundNode() method in FrameworkFrontEnd.java . . . 92
10.3.8 The connectToNodes() method in FrameworkFrontEnd.java . . . 93
10.3.9 The synchronizeGroups() method in FrameworkFrontEnd.java . . . 94
10.3.10 The sendTextPackage() method in FrameworkFrontEnd.java . . . 96
10.3.11 The sendDataPackage() method in BluetoothNetwork.java . . . 97
10.3.12 The processSendQueue() method in NodeConnection.java . . . 99
10.3.13 The processIncomingData() method in NodeConnection.java . . . 103
10.3.14 The notifyAboutReceivedTextPackage() method in FrameworkFrontEnd.java . . . 107

IV A Developers Guide to the Peer2Me Framework v2.0 **109**

11 Getting Started with the Peer2Me Framework v2.0 **111**
11.1 Peer2Me v2.0 Domain Concepts . . . 111
11.2 Required Resources . . . 112

12 Developing a Peer2Me v2.0 MIDlet **115**
12.1 Initiating the Framework . . . 115
12.2 Setting Up a Connection . . . 116
12.3 Sending a Data Package . . . 116
12.4 Using the Log . . . 117

13 Deploying a Peer2Me MIDlet **119**
13.1 Creating a MIDlet package . . . 119
13.2 How to run a MIDlet . . . 119

V Peer2Me v1.0 vs. Peer2Me v2.0 **121**

14 Comparison of Framework Functionality **123**
14.1 Peer2Me v2.0 Functionality . . . 123
14.1.1 Pure peer-to-peer computing . . . 124
14.1.2 Sending text . . . 124
14.1.3 Sending files . . . 124

14.1.4 Logging . . . 125
14.1.5 Detection of lost nodes . . . 125
14.1.6 Clean exit . . . 125

15 Comparison of Code Structure **127**
15.1 Code Samples . . . 127
15.1.1 Initiation of the framework . . . 128
15.1.2 Variable Comments . . . 129
15.1.3 Javadoc Comments . . . 130
15.1.4 Method and variable names . . . 131
15.1.5 Tidy Code . . . 132
15.2 Summary . . . 133

16 Comparison of Framework Properties **135**
16.1 The Goal/Question/Metric Approach . . . 135
16.2 Comparing the Framework Properties . . . 136

VI Evaluation **139**

17 Technology Evaluation **141**
17.1 Mobile Phones and J2ME . . . 141
17.1.1 General Evaluation of Mobile Phones and J2ME . . . 141
17.1.2 Strengths of Mobile Phones and J2ME . . . 142
17.1.3 Weaknesses of Mobile Phones and J2ME . . . 142
17.2 Bluetooth . . . 143
17.2.1 General Evaluation of Bluetooth . . . 143
17.2.2 Strengths of Bluetooth . . . 143
17.2.3 Weaknesses of Bluetooth . . . 144
17.3 Development Tools . . . 144
17.3.1 General Evaluation of the Development Tools . . . 145
17.3.2 Strengths of the Development Tools . . . 145
17.3.3 Weaknesses of the Development Tools . . . 145

18 Evaluating the Redesign **147**
18.1 General Evaluation of the Peer2Me Framework v2.0 . . . 147
18.2 Strengths of the Peer2Me Framework v2.0 . . . 148
18.3 Weaknesses of the Peer2Me Framework v2.0 . . . 148

19 GQM; Analysis and Interpretation **151**
19.1 Evaluating Goal 1 . . . 151
19.2 Evaluating Goal 2 . . . 153

VII Conclusion **155**

20 Conclusion **157**

21 Further Work **159**
21.1 Short Term Goals . . . 159
21.2 Long Term Goals . . . 159

VIII Appendices **161**

A Glossary **163**
Glossary . . . 163

B Demo MIDlets **165**
B.1 Peer2MeDemoMIDlet . . . 165
B.2 Peer2Messenger . . . 183

C Peer2Me v2.0 Javadoc **197**
C.1 Package peer2me.domain . . . 197
C.1.1 Class DataPackage . . . 198
C.1.2 Class FilePackage . . . 200
C.1.3 Class Group . . . 202
C.1.4 Class GroupSyncPackage . . . 205
C.1.5 Class Node . . . 206
C.1.6 Class TextPackage . . . 209
C.2 Package peer2me.framework . . . 211
C.2.1 Interface Framework . . . 211
C.2.2 Interface FrameworkListener . . . 214
C.2.3 Class FrameworkFrontEnd . . . 216
C.3 Package peer2me.network.bluetooth . . . 221
C.3.1 Interface BluetoothServiceDiscoveryListener . . . 221
C.3.2 Class BluetoothNetwork . . . 222
C.3.3 Class BluetoothServiceDiscovery . . . 225
C.4 Package peer2me.network . . . 228
C.4.1 Class ConnectionListener . . . 228
C.4.2 Class Network . . . 229
C.4.3 Class NodeConnection . . . 233
C.5 Package peer2me.util . . . 236
C.5.1 Class ASCIIToHexConvert . . . 236
C.5.2 Class FileHandler . . . 237
C.5.3 Class Log . . . 239

D Peer2Me v2.0 Source code **243**
D.1 Package peer2me.framework . . . 243
D.1.1 Interface Framework . . . 243
D.1.2 Interface FrameworkListener . . . 246
D.1.3 Class FrameworkFrontEnd . . . 248
D.2 Package peer2me.domain . . . 258
D.2.1 Class DataPackage . . . 258
D.2.2 Class TextPackage . . . 262
D.2.3 Class FilePackage . . . 266
D.2.4 Class GroupSyncPackage . . . 270
D.2.5 Class Group . . . 274
D.2.6 Class Node . . . 278
D.3 Package peer2me.util . . . 282
D.3.1 Class Log . . . 282
D.3.2 Class FileHandler . . . 287
D.3.3 Class ASCIIToHexConvert . . . 296
D.4 Package peer2me.network . . . 298
D.4.1 Class Network . . . 298

D.4.2 Class NodeConnection 304
D.4.3 Class ConnectionListener 318
D.5 Package peer2me.network.bluetooth 322
D.5.1 Class BluetoothNetwork 322
D.5.2 Interface BluetoothServiceDiscoveryListener 331
D.5.3 Class BluetoothServiceDiscovery 332
Bibliography 340

List of Tables

2.1 The Disciplines and artifacts of the RUP lifecycle 15

6.1 Peer2Me v1.0 Functional Requirements 43
6.2 Peer2Me v1.0 Non-Functional Properties 43

8.1 Peer2Me Functional Requirements 54
8.2 Use Case 1 57
8.3 Use Case 2 58
8.4 Use Case 3 59
8.5 Use Case 4 60
8.6 Use Case 5 61
8.7 Use Case 6 62
8.8 Use Case 7 63
8.9 Peer2Me v1.0 Non-Functional Requirements 64

16.1 Framework Properties 137

List of Figures

2.1 Iterative Development in the RUP . . . 13
2.2 GQM Hierarchical structure . . . 16
2.3 Overview of the experiment process . . . 18

3.1 Classifying groupware with the time-place matrix . . . 23
3.2 Taxonomy of computer systems . . . 25
3.3 Pure peer-to-peer model . . . 26
3.4 Hybrid peer-to-peer model . . . 27
3.5 A digital sphere around the user . . . 28

5.1 J2ME Architecture . . . 34
5.2 The Bluetooth logo . . . 36
5.3 A piconet comprising four nodes . . . 38
5.4 A scatternet comprising three piconets . . . 39

6.1 Architectural overview of the Peer2Me framework . . . 44
6.2 A logical view showing the main packages in the Peer2Me framework v1.0 . . . 45

8.1 Use Case 0 model . . . 56

9.1 Module Decomposition View . . . 70
9.2 Peer2Me Architecture . . . 72
9.3 Class Diagram . . . 73

10.1 Sequence Diagram . . . 80

11.1 Peer2Me conceptual model . . . 112

Part I

Introduction

CHAPTER 1

Introduction

SSB, Statistics Norway's Statistical Yearbook 2005 [37] shows us that the number of mobile telephony subscriptions (4 716 090 in 2004) is steadily increasing, and hence is the number of mobile phones also growing. New mobile phone models pours out on the marked and the technology that was considered "high end" last year have become "low end" already. Recreation and entertainment is the largest area of innovation at the moment, but also mobile ad-hoc communication and collaboration are important features of these new devices.

Today Java 2 Platform, Micro Edition (J2ME) [26] is by far the most common used programming platform for mobile devices like mobile phones. The technology is supported widely among electronics manufacturers and software developers. With the constant development and implementation of new technology leading to more powerful and better equipped phones, the potential for developing more "useful" applications grows. The introduction of means of communication directly between phones, such as Bluetooth, adds a new dimension to this type of applications. This gives the software engineers the ability to develop systems that allows collaboration and exchange of information between users.

Michael Sars Norum and Carl-Henrik Wolf Lund developed the Peer2Me framework as a part of their depth study [30] and master thesis [31]. The J2ME-based framework aims to aid the development of collaborative mobile applications using ad-hoc networking. Peer2Me gives developers the ability to program applications for mobile devices without considering the underlying network technology and how data is sent over this network.

1.1 Motivation

After working with the Peer2Me framework for a whole semester through our depth study [39], we have experienced some issues and come up with quite a few suggestions for improvements. We are under the impression that the Peer2Me framework have to undergo a serious revision to advance into a product with a practical application among developers. The revision involves several areas and properties of the Peer2Me:

Easy to learn The framework has to become easier to start using for the developers. Many programmers will experience the framework somewhat cumbersome to start using and even continue using after the initial learning phase.

Lightweight The framework has to keep and preferably improve it's lightweight structure. Considering the range of devices this framework is intended for this is important.

Peer-to-peer computing The Peer2Me framework is, as the name implies, based on the concept of peer-to-peer computing. But the current version of the system does not support this to full extent.

Transfer data To have more practical applications, the Peer2Me framework needs to support sending of more than just text, e.g. files.

Separate network layer To be totally network independent, the framework has to have a completely separate network layer architecture. This is not the case in the original framework.

The wish to improve these issues mentioned above is what motivates us to undertake this project. We believe that if these initial problems of the framework are overcome, it can turn out to be an interesting system with many employments.

1.2 Problem Definition

The title of this master thesis is:

> **Redesign and optimalization of the Peer2Me Framework**
> - A framework for developing Applications supporting mobile collaboration using J2ME

The main goal of the project is to redesign the Peer2Me framework, originally developed by Sars Norum and Wolf Lund in their master thesis from spring 2005. Peer2Me is a framework for developing mobile collaborative applications on mobile phones utilizing Personal Area Networks (PANs). The framework is developed using J2ME technology and currently supports Bluetooth communication.

The Peer2Me framework has thoroughly been tested and analyzed in two separate depth study projects, and several possible improvements have been discovered. This includes improving the architecture, simplifying the interface presented to the developer and decrease its footprint.

To decide whether the improvements have been successful, the original and redesigned Peer2Me frameworks

will be compared with regard to size, complexity, architecture and user experience for the developers. The results will be analyzed to determine the quality of the redesigned framework.

1.3 Project context and limitation of scope

As our depth study from 2005 [39] this master thesis project is a part of the Mobile Work Across Heterogeneous Systems (MOWAHS) Project. As mentioned in the depth study, MOWAHS is a basic research project performed in cooperation between the IDI's groups for software engineering and database technology. The project is supported by the Norwegian Research Council through the IKT-2010 program with a budget of 5 million NOK over four years.

The founders of MOWAHS have stated three goals for the project:

G1) Helping to understand and to continuously assess and improve work processes in virtual organizations.

G2) Providing a flexible, common work environment to execute and share real work processes and their artifacts, applicable on a variety of electronic devices (from big servers to small PDAs).

G3) Disseminating the results to colleagues, students, companies, and the community at large.

The original developers of the Peer2Me framework wrote in their depth study [30], that the creation of the framework was contributing to the second of the three MOWAHS goals. As we are improving and optimizing the Peer2Me framework, this project supports the second goal as well. Our redesign will take the Peer2Me framework further towards fulfilling the goal.

As in the depth study this project is more of a software engineering character, rather than focusing on collaborative work. The collaborative concept is described in the section about Computer Supported Collaborative Work (CSCW) in Chapter 3, Central Concepts.

1.4 Reader's Guide

The reader's guide is meant to describe the different parts of this document, as it is rather large and divided into both parts and chapters. We list each chapter with a short summary of the content.

Part I Introduction This part contains the introduction and the research questions and method.

- **Chapter 1 Introduction** The first chapter consists of the motivation, the definition of the problem, project context, limitation of scope and this reader's guide.
- **Chapter 2 Research Questions and Method** The motivation and problem definition gives us some research questions we need to answer. To do this we need a research method and a development method.

Part II Prestudy In this part the prestudy is described.

Chapter 3 Central Concepts Describes the central concepts that affect the project.

Chapter 4 Software Architecture This chapter contains an introduction to quality driven software architecture.

Chapter 5 Technology Here we will explain the latest aspects of the technology we use in the development face of the project.

Chapter 6 The Original Peer2Me Framework This chapter contains a short description of the first version of the Peer2Me framework and it's concepts.

Chapter 7 Related work Here we will look into related work in the research field.

Part III Redesigning the Peer2Me Framework v1.0 This covers the design and implementation of the redesigned Peer2Me framework as well as the elicitation of the requirements.

Chapter 8 Requirements Using Use Cases we will elicit the systems functional requirements. The non-functional and environmental requirements will be discussed as well.

Chapter 9 Design In this chapter the high level architecture of the system is described. We will also briefly explain the architectural patterns used in the design and give a short introduction to the individual classes of the framework.

Chapter 10 Implementation Here the implementation method and tools are described. Some code samples from the application is also provided.

Part IV A Developers Guide to the Peer2ME Framework v2.0 This part contains all the information a developer needs to start programming a Peer2Me application.

Chapter 11 Getting Started with the Peer2Me Framework v2.0 This chapter contains an introduction to the new version of the Peer2Me Framework.

Chapter 12 Developing a Peer2Me MIDlet Includes a description of how to develop a functioning Peer2Me MIDlet from scratch.

Chapter 13 Deploying a Peer2Me MIDlet Describes how the finished MIDlet could be deployed to a portable J2ME device.

Part V Peer2Me v1.0 vs. Peer2Me v2.0 Contains a comparison of the previous and the new Peer2Me version.

Chapter 14 Comparison of Framework Functionality This chapter contains a comparison of functionality found in Peer2Me v1.0 and Peer2Me v2.0.

Chapter 15 Comparison of Code Structure In this chapter we describe the improvements made to the code structure of the redesigned Peer2Me by comparing it to the structure of similar code found in the original framework.

Chapter 16 Comparison of Framework Properties In this chapter we compare the properties of Peer2Me v1.0 with the properties of Peer2Me v2.0. The properties we compare are; footprint of framework and MIDlets, size of interface, complexity and transfer rate.

Part VI Evaluation Based on the test result we will evaluate the Peer2Me framework and discus usability, strengths and weaknesses.

Chapter 17 Technology evaluation This chapter contains an evaluation of the technology used throughout this project. Strengths and weaknesses of the technologies are presented.

Chapter 18 Evaluating the Redesign Evaluating the redesign of the framework from a developer point of view.

Chapter 19 GQM; Analysis and interpretation This is the last phase of the Goal/Question/-Metric approach. It answers the questions raised and evaluates whether the goals are reached or not.

Part VII Conclusion The Conclusion of the project.

Chapter 20 Conclusion Here we summarize and find answers for our research questions.

Chapter 21 Further work If there is parts of our work we are not satisfied with we will suggest further work in this chapter.

Part VII Appendices The Bibliography and the Glossary.

1.4.1 Which Chapters to Read

Readers that just want to read about the main results of this report, we advice to read Part VI Evaluation, and the conclusion in Chapter 20.

If you want to read about the limitations of mobile phones and Bluetooth, please read Chapter 17 Technology evaluation.

For information about how the redesigned Peer2Me framework performed as a framework for developing applications for mobile collaborative work we advice you to read Chapter 18.

If you are a developer interested in how to develop a Peer2Me application, please read Part IV A Developers Guide to the Peer2Me Framework.

CHAPTER 2

Research Questions and Method

The questions that we hope to answer in our research and testing in this project are outlined in this chapter. We also describe the methodologies we choose to use to perform the research and to develop the test application. The last chapter is about the test environment we will carry out our tests in.

2.1 Research Questions

The original version of the Peer2Me framework has proven to be a useful tool for developers creating J2ME applications. However, some shortcomings has been discovered. This has resulted in several questions arising from the motivation and problem definition, and these questions are formalized in the following. This report will lead to the answers of these questions.

1. **Does a redesign of the Peer2Me framework improve developers ability to produce J2ME based applications for mobile collaboration?**

   * After we have completed our redesign of the Peer2Me framework, we will perform a comparison of the original and the redesigned framework to point out the differences and improvements we have achieved.

   (a) Is the documentation and the code, with regards to structure and comments, improved sufficiently to decrease the degree of difficulty developing a new application?

      * We will thoroughly compare the documentation and the code, with regards to structure and comments, of both the original and redesign versions of the framework.

   (b) Does the redesigned architecture increase the developers understanding of the framework's structure, and by this simplify the process of developing a working application?

    * The architectures of the original and the redesigned Peer2Me framework will be evaluated with regard to complexity and structure. To evaluate this we will use comprehension of the architecture, use of patterns and best practices and the coupling between the different modules of the framework as criteria.

2. **Does a redesign of the Peer2Me framework reduce the footprint and the complexity of the applications developed as well as the framework itself?**

    * The redesign will aim to improve the quality of both the framework and the finished applications with regards to size and structure. Data concerning this properties will be collected through a comparison of the original and redesigned framework.

    (a) Will the redesign of the architecture reduce the footprint of the framework?

        * By reducing the number of classes and relations in the framework the size will be reduced considerably.

    (b) Will improving the interface between the Peer2Me framework and the applications reduce the number of code lines required to develop a working application?

        * If the number of code lines are reduced by an improved architecture the development time as well as the footprint of the applications will be reduced. This enables the developer to create better applications more efficiently.

    (c) Will the redesign of the architecture reduce the coupling between the Peer2Me framework and the applications?

        * By reducing the number of relations connecting the framework and an application the complexity of development will be reduced.

3. **Will a redesign of the Peer2Me framework increase the performance and decrease the error rate of the applications developed?**

    * The redesign will address many of the problems found in Peer2Me applications developed upon the original version of the framework.

    (a) Does the redesigned Peer2Me framework perform better, with respect to transfer rate, than the original framework?

        * The original Peer2Me framework transfer rate was measured during the scenario testing of out depth study [39].

    (b) Does a revision of the code remove the errors experienced during testing of the original framework?

        * The original Peer2Me framework contains some errors that force the users to do operations in a certain order and to restart the application or the mobile phone.

    (c) Will the introduction of a system for logging the errors as they occur improve the developers ability to correct these errors?

        * If the errors of the application is written to a log the developers have a better chance of locating and correcting these errors.

2.2 Research Method

This chapter is a revision of the Research Method chapter found in our depth study [39]. The content and structure of the chapter is basically unchanged, but some sections have been altered to suite this project.

According to Basili in [4], software engineering:

> "...can be defined as the disciplined development and evolution of software systems based upon a set of principles, technologies, and processes."

The discipline of software engineering is fairly new in a scientific perspective. There is quite a lot of research going on in the field, but unlike other sciences the development of models for components like processes and resources have been neglected. How these models should be integrated, evaluated and used in projects are not satisfactory described either. Basili [4] describes three experimental models for software engineering research. The models are quite similar, but focus on different areas and are parts of two distinct paradigms; the scientific- and the analytical paradigms. The first consisting of the engineering approach and the empirical approach and the latter of the mathematical approach.

The three approaches in short:

The engineering approach (scientific) In this approach one have to perform iterations of observing the existing system, suggesting improvements and building and analyzing the new system. This continues until no more improvements can be found.

The approach is strictly evolutionary and implies access to existing models of processes, products and the environment in which the software is developed.

The empirical approach (scientific) Based on a model of the domain a set of statistical and qualitative methods are proposed. Then these models are applied to case studies, measured and analyzed, and the result is a validation of the model.

This distinct the approach from the previous one since a new model is proposed. It is also more reliable to validate the model through the use of case studies. This approach is widely used in all fields of research.

The mathematical approach (analytical) A formal theory or a set of axioms is presented, the theory are developed and a result is derived from it. It's preferable to have this results compared to empirical observations.

The two approaches of the scientific paradigm, engineering and empirical, will constitute the base for Parts III, V and VI of this project. The engineering approach will be utilized through the redesign of the Peer2Me framework described in Part III. In the comparison of the original and the redesigned Peer2Me framework outlined in part V, we will collect empirical data that will be used in the evaluation in Part VI.

2.2.1 The engineering approach

In Chapter 2.2 above we described the engineering approach as; observing, suggesting improvements, analyzing and building. The observation phase consists of the Prestudy, Part II, where we will look into central concepts, aspects about the Peer2ME framework that we discovered through the work performed in our depth study [39] and new technology as well.

Since the main focus of this project is to redesign and improve the Peer2Me framework we will incorporate the last three phases of the engineering approach into the requirements, design and implementation chapters of Part III, Redesigning the Peer2Me Framework v1.0.

When we stared the depth study we considered using the Rational Unified Process (RUP) [22] as a development process for the project. However, we soon realized that the RUP would be a bit to heavy weight for a project of this scale, so we opted for an adapted process still using the principles of UP[1] development. The iterative and incremental process of analyzing and building described in the Engineering approach, is a significant part of UP. The use of this approach turned out to be perfect for this type of project and we will continue to use it in our current process.

Prestudy

In our Prestudy we will look into and describe the central concepts that concerns our domain. We will also make a summary of and outline the essentials of the Peer2Me framework and review the newest technology in the field. This will give us an overview of what we have to work with and what challenges we have to overcome in our redesign.

The Unified Process

As described above we have chosen to continue with our adapted UP approach for this project, so we will keep many of the elements found in the RUP. In the following we describe the elements we kept and their role in the RUP.

In [22] the Rational Unified Process (RUP) is defined as:

> "...a **software development process** that is iterative, architecture-centric, and use-case-driven."

The process is intended to be tailored to select the most appropriate development processes for a software project and is based upon using proven techniques to develop software effectively. The RUP uses and iterative approach that consists of a sequence of incremental steps or iterations. Figure 2.1 shows how a typical iterative development with RUP is carried out. It starts with Business Modeling and Planning before it enters a loop of iterations. Each iteration consists, more or less, of this parts; Requirements elicitation, Analysis & Design, Implementation, Testing and Evaluation. Each iteration is based upon the work performed in the previous iterations and the result is one step closer to deployment.

[1] Unified Process

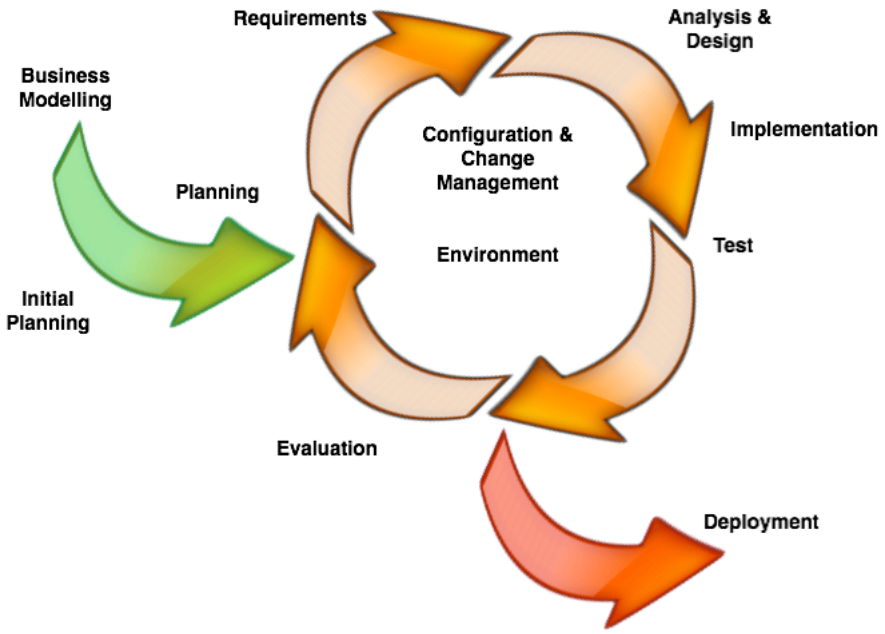

Figure 2.1: Iterative Development in the RUP

Here is a description of how we intend to perform the different tasks of the Iterative Development model during the development of our Peer2MeAnalyzer application.

* **Business Modeling and Planning** is covered by our prestudy where we investigate the Peer2Me framework and familiarize with the central concepts and new technology of the domain.

  The tasks in the iterative loop:

  - **Requirements** are specified through Use Cases and Use Case models. In addition we will specify non-functional requirements textually.
  - **Analysis and Design** will imply analysis of the elicited requirements and choosing a software architecture with the desired properties.
  - **Implementation** will be performed in parallel with some development testing using an iterative approach.

- In addition to **test** simultaneously with the implementation we will perform a more thorough test when the framework is ready to use. This is to ensure that it fulfils all the requirements and has a low fault rate. The test will be performed as a workshop with developers performing given tasks.
- **Evaluation** will be done at the end of each iteration to decide whether the framework has reach a satisfactory level of quality or if another iteration is necessary.
- **Configuration & Change Management** will be supported by the use of CVS[2] to keep track of the different versions of the application during development.
- The base of the development **environment** will be the Eclipse[3] software framework. In addition we will make use of the SUN Java Wireless Toolkit[4] for J2ME specific functionality. The documentation is written in LaTeX using a plugin for Eclipse.

* At **deployment** we will have the redesigned and improved the Peer2Me framework ready for developers to use.

The RUP architecture is divided into two dimensions, often presented in a diagram with the two dimensions along the vertical (Static) and horizontal (Dynamic) axis.

Dynamic structure This is the time dimension and describes the cycles, phases, iterations and milestones of the process. It illustrates the lifecycle of a project.

Static structure Describes the elements involved in the process and how they are grouped into process disciplines. The elements can be activities, disciplines, artifacts and roles.

We will now describe the four lifecycle phases of RUP.

Inception Phase This phase consists of understanding the scope of the project, build a business case and get stakeholder approval. This leads up to the Lifecycle Objective Milestone.

Elaboration Phase Here one will try to reduce major technical risks, create the outline of an architecture and find out what is required to build the system. In this phase the milestone is the Lifecycle Architecture.

Construction Phase To build the first operational version of the product is the goal of this phase and it ends in the Initial Operational Capability Milestone.

Transition Phase Lastly the final version is built and delivered to the customer. The milestone of this phase is Product Release.

Each of this phases contains one or more iterations and there will be as many iterations as it takes to fulfill the objectives of the phase. In Table 2.1 we have described what disciplines and artifacts we worked with in each of the four phases of the lifecycle.

[2] Concurrent Versions System
[3] www.eclipse.org
[4] http://java.sun.com/products/sjwtoolkit/index.html

Disciplines	**Artifacts**	**Inception**	**Elaboration**	**Construction**	Transition
Requirements	Use Cases	Start	Refine		
	Non-functional Requirements	Start	Refine	Refine	
	Environmental Requirements	Start	Refine	Refine	
	Glossary	Start	Refine	Refine	
Analysis and Design	Model Decomposition View		Start	Refine	
	Class/Uses View		Start	Refine	
Implementation	Source Code		Start	Refine	Refine
Test	GQM		Start	Refine	
	Scenarios		Start	Refine	
Evaluation	Technology	Start	Refine	Refine	Refine
	Peer2Me framework		Start	Refine	Refine
	Peer2MeAnalyzer			Start	Refine

Table 2.1: The Disciplines and artifacts of the RUP lifecycle

2.2.2 The empirical approach

In "An Empirical Methodology for Introducing Software Processes"[34], Shull et al. describes some important aspects of empirical studies in software engineering.

First and foremost it's important to separate between qualitative and quantitative data:

Quantitative data ...can be used for measuring a particular aspect of a process, e.g. "number of nodes detected". In other words this is numerical data that can be measurement and statistics.

Qualitative data ...is expressed in words and gives a richer understanding of the gathered information. This describes the perceived quality of the results.

Both data types are important in testing, since quantitative data is useful when performance is evaluated and qualitative data can say something about the usefulness and quality of a system.

The Research Questions in Chapter 2.1 are of both quantitative and qualitative form. The qualitative data will be collected through a comparison of the original and the redesigned Peer2Me framework. This comparison is found in Part V Peer2Me v1.0 vs. Peer2Me v2.0. To find answers for the quantitative related questions we will have to do empirical experiments in form of a comparison of the framework properties. This comparison is described in Chapter 16. To perform and document these experiments we choose to use of the Goal/Question/Metric paradigm as described in the following.

The Goal/Question/Metric paradigm

In this section we will describe the methodology we intend to use in our empirical study of the redesigned Peer2Me framework, and also how we will adapt the methodology to best fit our project.

The Goal/Question/Metrics (GQM) approach, proposed by Basili [41], assumes that an organization must fulfill three conditions to be able to measure results from experiments in a purposeful way:

1. Specify the goals for itself and its projects.
2. Trace those goals to the data that is intended to define those goals operationally.
3. Provide a framework for interpreting the data with respect to the stated goals.

The application of the GQM results in a measurement model specification targeting a set of issues and rules. These are issues and rules for the interpretation of measured data. The resulting model (see Figure 2.2) includes these three levels:

1. **Conceptual level** This is the Goal part of the model. A goal is defined for an object, for a variety of reasons, with respect to various models of quality, from various points of view, relative to a particular environment. Object of measurement are products, processes, and resources.
2. **Operational level** The Question part of the model. A set of questions is used to characterize the way the assessment/achievement of a specific goal is going to be performed based on some characterization model. Questions try to characterize the objects of measurement (product, process and resource) with respect to a selected quality issue and to determine its quality from the selected viewpoint.
3. **Quantitative level** The Metric. A set of data is associated with every question in order to answer it in a quantitative way (either objectively or subjectively).

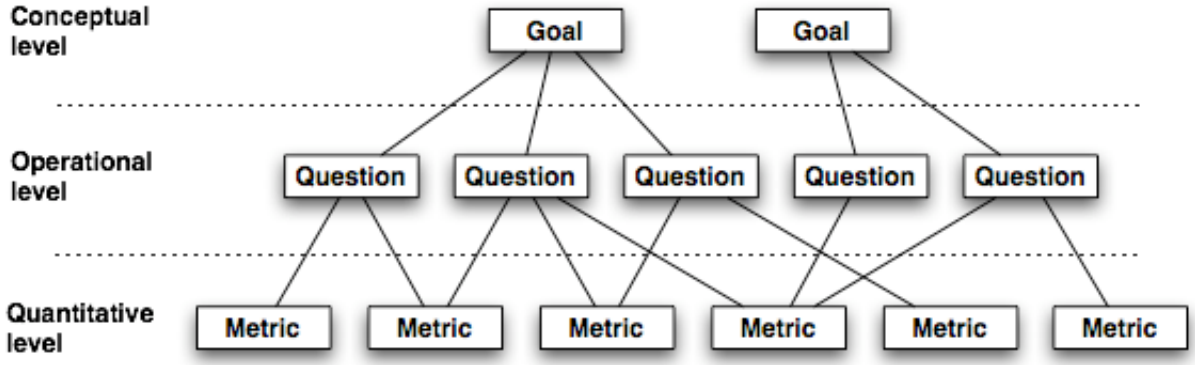

Figure 2.2: GQM Hierarchical structure

The definition of the Goal definition template can be found in [43]. The template is used to make sure that none of the important aspects of the goal are forgotten when the goal is defined in the description. The template is:

> Analyze **Object(s)** of study
> for the purpose of **Purpose**
> with respect to their **Quality** focus
> from the point of view of the **Perspective**
> **in the context of Context.**

In *Experimentation in software engineering: an introduction* [43], the main activities (see Figure 2.3) of the experimentation process are defined as:

Definition is the first step. Here the experiment is defined in terms of problem, objective and goals.

Planning is the next step, where the design of the experiment is laid down. The instrumentation is considered and the threats are evaluated.

Operation of the experiments consists of measurement collection.

Analysis and interpretation In this activity the measurements from Operation is analyzed and evaluated.

Presentation and package This is the final results.

The model is not to be interpreted as a "true" waterfall model because one activity is not necessarily finished before the next activity is started.

We will make use of the GQM approach in part V, "Peer2Mev1.0 vs. Peer2Mev2.0", where we will describe how we perform empirical experiments in our framework properties comparison to gather quantitative data about the redesigned framework.

2.2.3 Evaluation

The Evaluation will be based on the results found when comparing the original and the redesigned framework (qualitative data), and the quantitative data gathered during the framework properties comparison.

Technology evaluation A subjective evaluation of the technologies used in the project. The evaluation will include; Mobile Phones and J2ME, Bluetooth and Development tools. For each category strengths and weaknesses will be presented as well as a general evaluation.

Evaluating the redesigned Peer2Me framework This chapter will be a summary of the results gathered from both the code structural and framework properties comparison.

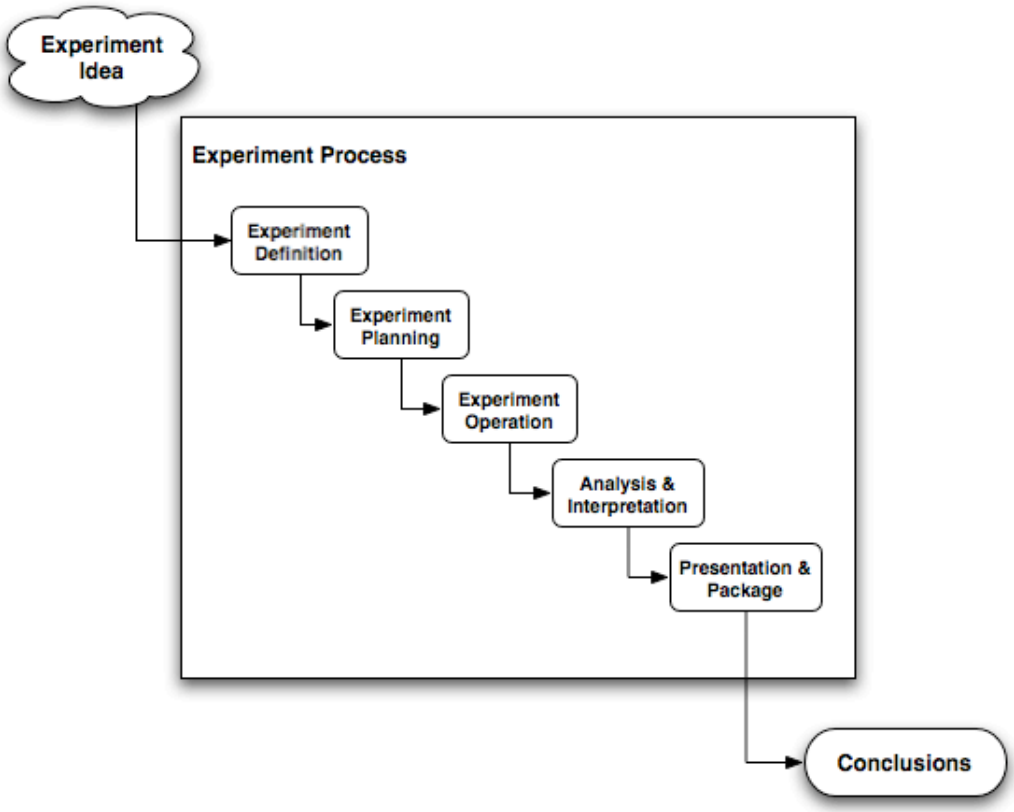

Figure 2.3: Overview of the experiment process

The Analysis and Interpretation phase of GQM The final phase of the GQM aims at answering the questions raised in Chapter 16.1. When all the answers are found one can conclude whether or not the goals are reached. The result of this can be used to determine if the redesigned Peer2Me framework fulfills it's purpose.

Part II

Prestudy

CHAPTER 3

Central Concepts

During the last few years' consumer electronics industries have continually introduced new mobile devices offering new features. A demand to be supported in the way we live and do things, has triggered an evolution which has provided a considerable amount of new benefits to today's users of mobile devices [28]. Among the exciting new features of today's mobile devices is the support for Peer-to-Peer (P2P) computing and Mobile Ad Hoc Networking (MANETs). Seen in relation with Computer Supported Cooperative Work (CSCW), these concepts have constituted a basis from which the Peer2Me framework has arisen.

In this chapter, we will present central concepts related to the Peer2Me framework. Since this master thesis basically is a continuance of our depth study, central concepts presented in this chapter will be the same as the ones presented in the Central Concepts chapter in our depth study report [39]. The content of the chapter is mainly unchanged, but some sections have been added to suite this project.

3.1 Cooperative Work

Cooperative work, literally, refers to the practice of people working together with commonly agreed upon goals. In other words, a cooperative work relationship is constituted by the fact that several workers are interdependent in their work. The cooperation is based upon interaction through changing the state of a common field of work.

Generally speaking, cooperative work relations are formed as a consequence of the limited capabilities of single individuals, that is, because the work could not be done otherwise, or at least not as efficiently, clever or quickly as if it was carried out by a single person. A general trend in modern work settings has been, and still is, that work becomes more and more complex. Demands for more complex products leads to demands for flexibility, shorter production time, increased quality etc. To accomplish such tasks, there is a need

for experts having different competence and backgrounds to work together towards the same goal. When the number of workers exceeds the limit of a few, and/or workers are located at different places working at different times, they will need a way to communicate and to coordinate their activities [6]. To develop successful methods/tools supporting collaboration between people, it is essential to understand the nature of human interaction. Human communication is very complex and involves a combination of:

Verbal communication: Communication between persons with the use of words. Includes both oral communication and written messages.

Nonverbal communication: Defined as communication between persons without using words. Most people use gestures and body language in addition to words when they speak. These gestures include acts such as pointing as well as using the hands and body to keep time with the rhythms of speech and emphasize certain words or phrases.

Formal communication: Communication between people in a formal setting, e.g. people communicating in well planned meetings.

Informal communication: Communication between people in an informal setting, e.g. people stopping and communicating as they accidentally meet in the corridors.

The combination of these aspects makes the natural human communication very complex.

3.1.1 Computer Supported Cooperative Work

Computer Supported Cooperative Work (CSCW) is a research field that focuses on how computer-based systems can support multiple people working on related tasks. 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 [14]. Since 1984, researchers have put a vast effort in into the area of CSCW. In spite of this, researchers and developers still struggle to come up with tools able to replace the value of being collocated while cooperating. To understand their problems, we have to consider the complexity of supporting collaboration in all the CSCW dimensions. One of the most cited classifications of CSCW systems, also called groupware, is the time-place matrix [20] shown in Figure 3.1.

In one dimension, the matrix distinguishes same time (synchronous) cooperative work from different time (asynchronous) cooperative work. In the other dimension, it distinguishes same place cooperative work from different place cooperative work. With this matrix, groupware systems can be classified by placing them in the quadrant(s) they support. The most complex challenges lies in developing groupware supporting workers who are not collocated (different places and/or different time). The reason for this is that the groupware becomes the only available communication channel between the workers. In these cases, the groupware has to address the complexity of human communication including support for awareness, verbal-, nonverbal-, formal- and informal communication. When it comes to workers who are located at the same place at the same time, other types of requirements will become relevant as these users might want to use computers to enrich the natural communication instead of replacing the natural communication between users [6] [15].

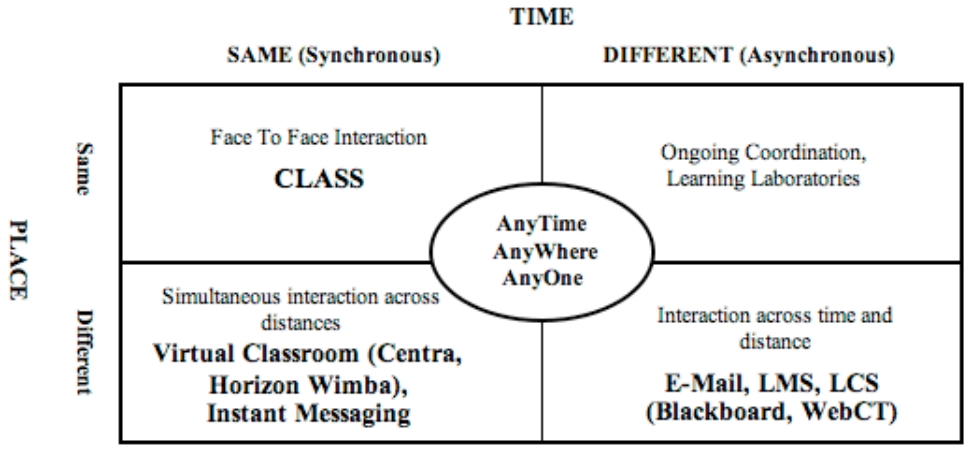

Figure 3.1: Classifying groupware with the time-place matrix

According to Lund and Norums master thesis [31], the Peer2Me framework covers both synchronous (same time) and asynchronous (different time) applications in the "same place" category. As the framework is based upon peer-to-peer networking (no infrastructure) and supports sharing of data (communication) between users, it is fairly obvious that it supports communication between users located at the same place at the same time. To understand how the framework can cover the same place, different time category, we have to focus on the users rather than the devices. Seen from the device's perspective, the framework requires the devices to be located at the same place at the same time for data exchanging to happened. But seen from the user's perspective, users are allowed to communicate asynchronously because sending of data does not require the recipient to be present. However, our opinion is that the Peer2Me framework mainly addresses the same place same time quadrant. This because the devices used in peer-to-peer networking usually are personal property that users constantly carry around with them.

3.1.2 Mobile Computer Supported Cooperative Work

Mobile Computer Supported Cooperative Work (MCSCW) is, as the name indicates, a research field that focuses on how mobile computers can support multiple people working on related tasks. The last years progress in technology when it comes to mobile devices has been amazing and the possibilities for using computers to support cooperative work is no longer limited to stationary devices. The extreme mobility and support for different kinds of wireless networks allows users to collaborate whenever they want, wherever they want. This includes both planned communication and ad hoc communication. Central concepts within MCSCW, such as mobile ad hoc networking and different types of wireless networks are thoroughly treated in respectively Chapter 3.4 and Chapter 3.2.

3.2 Wireless Networking

In general, wireless networking refers to the use of infrared or radio frequency signals to share information and resources between devices. Rapid advances in wireless technologies have changed the wireless communication landscape dramatically during the past years. Today, we have many types of wireless networks. These types can mainly be divided into two categories; telephone networks and computer networks. In the following we will focus on computer networks, and try to classify and describe their characteristics.

3.2.1 Wireless Wide Area Networks (WWANs)

Wireless Wide Area Networks, from now on referred to as WWANs, are infrastructure-based networks built up by a set of base stations broadcasting radio signals to mobile users. WWANs addresses the need to stay connected while traveling across large geographical areas as the reach range of the wireless signals are wide. Connections can typically be made over cities or even countries. Today, the networks we use to make such connections are actually cellular telephone networks. Networks like GSM (2G) and UMTS (3G) enable wireless computer connectivity almost worldwide and support transmission of both speach and raw data. [2] [3]

3.2.2 Wireless Metropolitan Area Networks (WMANs)

Wireless Metropolitan Area Networks, hereby abbreviated as WMANs, are also infrastructure-based networks built up by a set of base stations. These networks connect users within metropolitan areas such as multiple buildings on a university campus or multiple office buildings. WMANs can be realized by a number of interconnected WiFi transmitters located in a way that covers the desirable area with radio signals. Otherwise it can be realized by using WiMAX that provides wireless coverage over many square kilometers, much greater than WiFi. WiMAX has the potential to allow wireless mobility over an entire metropolitan area. [3]

3.2.3 Wireless Local Area Networks (WLANs)

Wireless Local Area Networks (WLANs) are networks, which allows users to establish wireless connections within local areas as e.g. buildings. WLANs can operate in both infrastructure-based and ad hoc mode. In the infrastructure mode, wireless stations connect to wireless access points that define a finite region of coverage. These access points bridges the wireless stations and the existing network backbone. The other alternative, the ad hoc mode, let users connect to each other without having a fixed infrastructure with access points. Instead the wireless stations connect to each other directly. This mode is only supported within very a limited area such as a room. Examples of typical WLAN protocols include the IEE 802.11 series (a,b,g), HomeRF and HiperLAN2. Today, the leading WLAN protocol in the consumer market is the IEEE 802.11g, which has a theoretical maximum data rate of 54Mbps. [2] [3]

3.2.4 Wireless Personal Area Networks (WPANs)

Wireless Personal Area Networks (WPANs) are networks connecting users within a personal operating space, typically supporting up to a ten meter range. This is the type of network the Peer2Me framework addresses. The emphasis is on instant connectivity between devices that manage personal data or which facilitate data

sharing between small groups of individuals. An example might be spontaneous sharing of documents and music files between two or more individuals. Another example might be synchronizing data (e.g. a calendar) between a mobile phone and a computer. The nature of these types of data sharing scenarios is that they are ad hoc and often spontaneous. The most relevant technologies suitable for WPANs today are Bluetooth, ZigBee and infrared light. Whereas infrared light demands clear vision between two peers and a maximum range (distance) of about one meter, Bluetooth is often more suitable due to the use of radio waves instead of light as transmission medium. Bluetooth uses the unregulated 2.4GHz band, has a maximum data rate of 1Mbps and a signal range of about ten meters. As the Peer2Me framework addresses WPANs and per today only supports Bluetooth, we will write more about mobile ad hoc networking and the Bluetooth technology in the following sections. [2] [3]

3.3 Peer-to-Peer Computing

The term "peer-to-peer computing", refers to the use of computer networks that relies in the computing power and bandwidth of the participants (peers) in the network rather than fixed servers offering resources and services. Peer-to-peer is all about sharing; giving to, and obtaining from a peer community. Peers typically depend on each other for getting computing resources and information, which are essential for the system as a whole. Each peer gives some resources and obtains other resources in return [29].

In Figure 3.2 from [29], Milojicic et al. show how all computer systems can be classified as centralized or distributed. While centralized systems represent single unit solutions, distributed systems consist of components located at networked computers which communicate and coordinate their actions by passing messages. Distributed systems can further be classified into a client-server model or a peer-to-peer model. The difference between these models is that the central unit in the client-server model is a server providing all services and resources, while the peer-to-peer model has no central unit. Each peer gives some resources and obtains other resources in return [29].

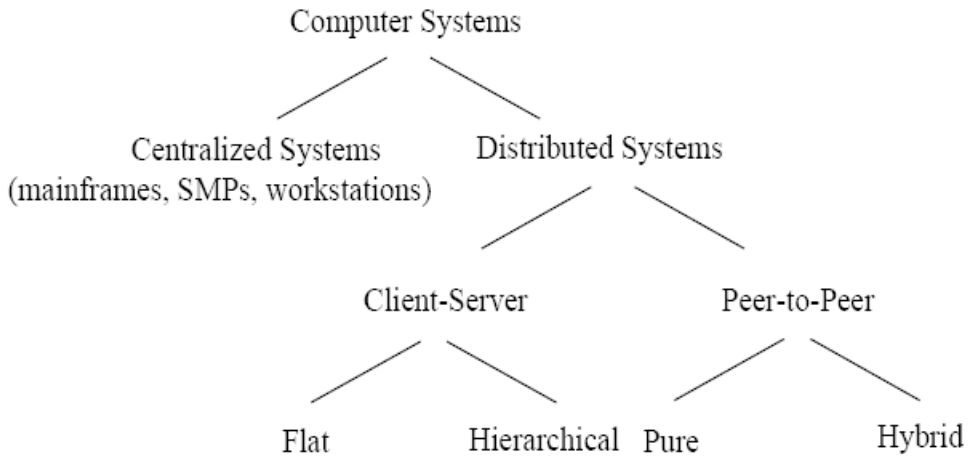

Figure 3.2: Taxonomy of computer systems

3.3.1 Pure or hybrid peer-to-peer

The peer-to-peer model can either be pure or hybrid. According to Lund and Norum [31], the Peer2Me framework they developed offers a combination of a pure and hybrid P2P architecture. In a pure P2P model it does not exist any central unit (server) responsible for managing or coordinating the services and the resources among the peers in the network. All peers are equal and have the same responsibility in the network. This pure P2P model allows peers to join and leave the network as they wish without affecting the connection between other peers. An example of a system using the pure P2P model is Gnutella, a system that offers peer-to-peer sharing of data between computers [29]. The pure P2P model is viewed in Figure 3.3.

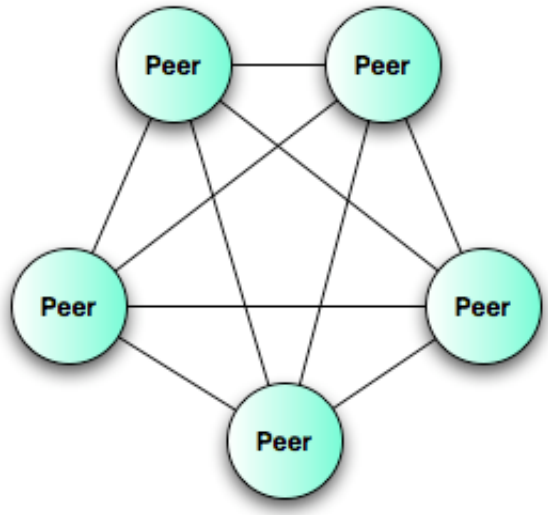

Figure 3.3: Pure peer-to-peer model

In a hybrid P2P model there are also connections between each peer, but a central unit (server) provides certain services to the peers. Figure 3.4 contains the P2P hybrid model. In this model, the peers first contact a server to obtain meta-information, such as the identity of the peer on which some information is stored, or to verify access to a specific peer. From then on, the communication between the peers is carried out. Examples of computer systems using hybrid P2P are file sharing systems such as Napster and iMesh [29].

Selecting a peer-to-peer architecture is often driven by goals such as cost reduction, need for improved performance, improved scalability, dynamism and ad hoc communication. As earlier mentioned in this chapter, all peers in a peer-to-peer network provide resources, including bandwidth, storage space, and computing power. Thus, as nodes arrive and demand on the system increases, the total capacity of the system also increases. This is not true in a client-server model with a fixed set of servers, in which adding more clients could mean slower data transfer for all users. The distributed nature of peer-to-peer networks also increases robustness in case of failures by replicating data over multiple peers and enabling the peers to

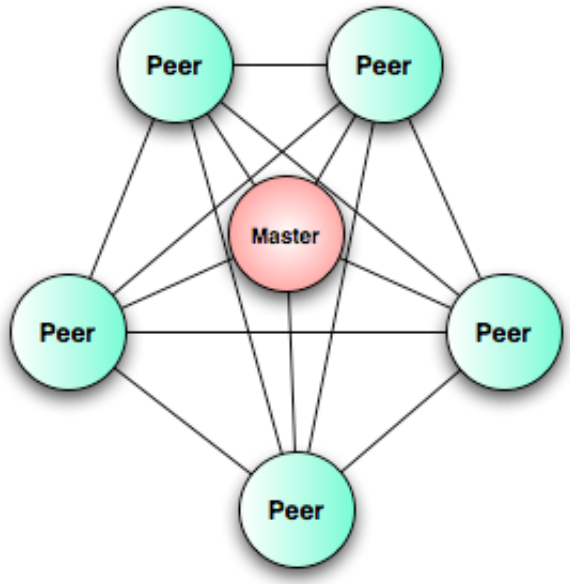

Figure 3.4: Hybrid peer-to-peer model

find data without relying on a centralized server. With a pure peer-to-peer architecture, there is no single point of failure in the system [29].

P2P networks are typically used for connecting nodes via largely ad hoc connections. Such networks are useful for many purposes. Sharing content files, audio files, video files or anything in digital format is very common. The P2P model is also widely used within instant messaging applications and is well suited for real time communication since it does not rely on a central server to collect and relay data [29].

3.4 Mobile Ad Hoc Networking

Mobile ad hoc networks, abbreviated MANETs, are networks formed dynamically by an autonomous system of mobile nodes that are connected via wireless links without using an existing network infrastructure. MANETs allows users to connect spontaneously with other users within the range of the wireless network signals [3]. This signal range can be seen on as a digital sphere surrounding the user (see Figure 3.5). When a user carrying a mobile device acting as a node moves out of the sphere, the connection between this node and the nodes inside the sphere is broken.

No infrastructure needed and quick deployment make mobile ad hoc networks perfect for supporting ad hoc communication between people and very suitable for emergency situations like natural or human induced disasters, military conflicts, emergency medical situations etc.

Network technologies supporting mobile ad hoc networking can mainly be found within the Wireless Personal

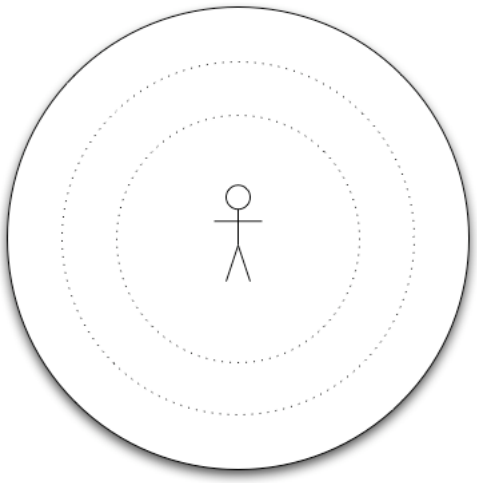

Figure 3.5: A digital sphere around the user

Area Network (WPAN) category described in Chapter 3.2.4. Examples of WPAN technologies would be Bluetooth, ZigBee and infrared light. A WPAN can either be a piconet or a scatternet. As the Peer2Me framework for the time being only supports Bluetooth, we will focus on Bluetooth while explaining the differences between a piconet and a scatternet. The explanation can be read in Chapter 5.2.

CHAPTER 4

Software Architecture

This chapter contains an introduction to software architecture. First, we will try to explain what software architecture is, and then present factors that are motivating when creating an architecture. The theory in this chapter will constitute a basis for understanding the architectural decisions we present in Chapter 9, Design.

4.1 What is Software Architecture?

An architectural view of a system is abstract and distills away details of implementation. The focus is on the behavior and interaction of the system elements. In *Software Architecture in Practice* [5], the following definition of software architecture is provided:

> ”The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.”

Architecture is a crucial part of the software design process. This because deciding the software architecture is the first step toward designing a system with the desired qualities and properties. Depending on the desired qualities, different architectural patterns can be used to achieve these goals. Architectural patterns can be thought of as general repeatable solutions to common problems, describing elements and relation types together with a set of constraints on how they may be used. A common architectural pattern is the client-server pattern. Here, client and server are element types, and the relation is described by the protocol they use to communicate [5] [27]. A description of the patterns used in the design of the Peer2Me framework we are about to present in Part III, Redesigning the Peer2Me Framework v1.0, can be read in Chapter 9.3, Design Patterns.

4.2 Creating an Architecture

There is no such thing as an inherently good or bad architecture. Architectures are either more or less fit. An architecture is the result of several business and technical decisions. Usually, several people are interested in the construction of a software system. These people are called stakeholders, and often includes both the project manager, the developers, the maintainers, the customer and the end users. As these stakeholders have different concerns depending on which properties and qualities they consider most important, they will try to influence the decisions taken in the design process in a way that protects their own interests. Besides influence by stakeholders, architecture can also be influenced by the developing organization, by the background and experience of the architects, and by the technical environment. If the organization and/or the developers have positive experiences using a specific architecture, chances are good they will choose the same architecture again.

In *Software Architecture in Practice* [5], some general rules of thumb to follow when designing an architecture are presented. The essence of these recommendations are presented in the following.

Process recommendations:

* The architecture should be developed by a single architect or a small group of architects with a leader.
* The architect should have the functional requirements that the system (and architecture) must satisfy.
* The architecture should be well documented in several views.
* The architecture should be reviewed by all stakeholders.
* The architecture should be analyzed and evaluated in an early phase, before it is too late to change it.
* The implementation of the architecture should be incremental, starting with the creation of a "skeleton", and adding functionality through several increments.

Structural recommendations:

* The architecture should consist of well defined modules built on the principles of information hiding and separation of concerns.
* Each module should have a well defined interface, hiding changes and allowing developers to work independently of each other.
* Quality attributes should be achieved using known tactics.
* The architecture should not depend on a specific version of a tool.
* Modules producing data and modules consuming data should be separated.

4.3 Quality Attributes

Achievement of desired quality attributes is critical to the success of a system. Desired qualities can be extracted from both system requirements and the interests of the stakeholders. Achieving quality attributes does not depend on the design alone, nor the implementation or the deployment, but the entire development process. It is however critical to the realization of many qualities that they already become designed in at the architectural level. There are usually more than one quality attribute involved in a system. A set of well-known quality attributes are listed in the book *Software Architecture in Practice* [5] and we will present these attributes in the following where we also mention some architectural tactics that can be used to achieve them.

4.3.1 Usability

The definition of Usability:

> ”Usability is concerned with how easy it is for the user to accomplish a desired task and the kind of user support the system provides.”

Usability is usually desirable from both a developer/maintainer perspective and an end-user perspective. Tactics to achieve good usability from a developers/maintainers perspective, are typically separation of user interfaces from the rest of the application and to provide a solid, easily understood documentation of the architecture and the source code. An end-user would typically focus on the quality of the feedback given by the system, giving information about what the system is doing. To provide good usability, the system should support user initiative based on the feedback, e.g. choose ”cancel” or ”undo” [5].

4.3.2 Performance

Performance:

> ”Performance is about timing. Events (interrupts, messages, requests from users, or the passage of time) occur, and the system must respond to them.”

To achieve high performance, decreasing resource demand and managing resources effectively is crucial. Optimizing algorithms, reducing computational overhead and reducing the number of events processed will contribute to decrease the resources demanded of a system. Other tactics increasing the performance are introduction of concurrency (threading), use of caching, and an increase of available resources (e.g. faster CPU, more RAM, faster network etc.) [5].

4.3.3 Modifiability

A definition of Modifiability:

> ”Modifiability is about the cost of change. It brings up two concerns; What can change (the artifact)? - When is the change made and who makes it (the environment).”

The goal of modifiability tactics is to control the time and cost to implement, test, and deploy changes. To achieve this goal semantic coherence between modules should be maintained and modules should be generalized. To prevent ripple effects when modifying modules in a system, information should be hided within the modules, and communication paths between modules reduced to a minimum. Other tactics contributing to great modifiability are use of configuration files and polymorphism [5].

4.3.4 Availability

This is said about Availability:

> ”All approaches to maintaining availability involve some type of redundancy, some type of health monitoring to detect a failure, and some type of recovery when a failure is detected.”

To achieve good availability, tactics detecting faults, recovering from faults and tactics that prevents failures are needed. Three widely used tactics for detecting faults are ping/echo, heartbeat and exceptions. To recover from faults, tactics like voting, use of redundancy and support for rollback could be used. Fault prevention can be achieved by using transactions, a process monitor and/or removal from service [5].

4.3.5 Security

Tactics for achieving security can be divided into three different categories; resisting attacks, detecting attacks and recovering from attacks. In *Software Architecture in Practice* [5] we find this analogy:

> ”Putting a lock on your door is a form of resisting an attack, having a motion sensor inside of your house is a form of detecting an attack, and having insurance is a form of recovering from an attack.”

To resist attacks, tactics like authentication and authorization of users could be used. Maintaining data confidentiality and integrity, and limit exposure and access are also well-known tactics to achieve better security. To detect attacks, systems should have an intrusion detection system consisting of sensors able to detect attacks. Recovering from attacks could be made possible by using e.g. redundancy and/or support for rollback [5].

4.3.6 Testability

The goal of Testability as a quality attribute is:

> ”Allow easier testing when an increment of software development is completed.”

Being able to manage input and output data is essential to achieve good testability. A well-known tactic is Record/playback where both input and output data could be recorded and compared each time the system is tested. Another tactic is to separate interface from implementation. This allows substitution of implementations and could be very useful [5].

CHAPTER 5

Technology

This chapter reviews and updates the technology review performed in our depth study [39]. This includes Java 2 Micro Edition which is the development platform the framework is built upon, and Bluetooth which is the wireless network medium the framework supports.

5.1 Java2 Micro Edition

Java 2 Micro Edition (J2ME) is Sun Microsystems' contribution to small mobile devices with limited CPU power, memory size and storage capacity. J2ME was introduced in June 1999, and is basically a platform which provides a robust, flexible environment for applications running on mobile phones, PDAs and other mobile devices [17]. The platform delivers the power and benefits of Java technology, and includes a broad range of built-in network protocols. The J2ME platform is supported by leading electronics vendors and used by companies all over the world. Today the platform is deployed on millions of mobile devices. Due to the diversity among devices, the J2ME architecture also comprises a variety of optional packages that can be added and used to construct a runtime environment that perfectly fits the requirements of a particular assortment of devices [26]. Such optional packages can for example add support for database connectivity, wireless messaging, multimedia, Bluetooth, or web services. Because the packages are modular, developers can avoid carrying the overhead of unnecessary functionality by including only the packages an application really needs [26]. The Peer2Me framework [31] is currently implemented with a Bluetooth [16] network module, and uses an optional package called JSR-82 which adds support for Bluetooth to the J2ME platform.

5.1.1 J2ME Architecture

The J2ME architecture is composed of three scalable layers; Java Virtual Machine (JVM), Configurations, and Profiles [17]. The architecture is viewed in Figure 5.1.

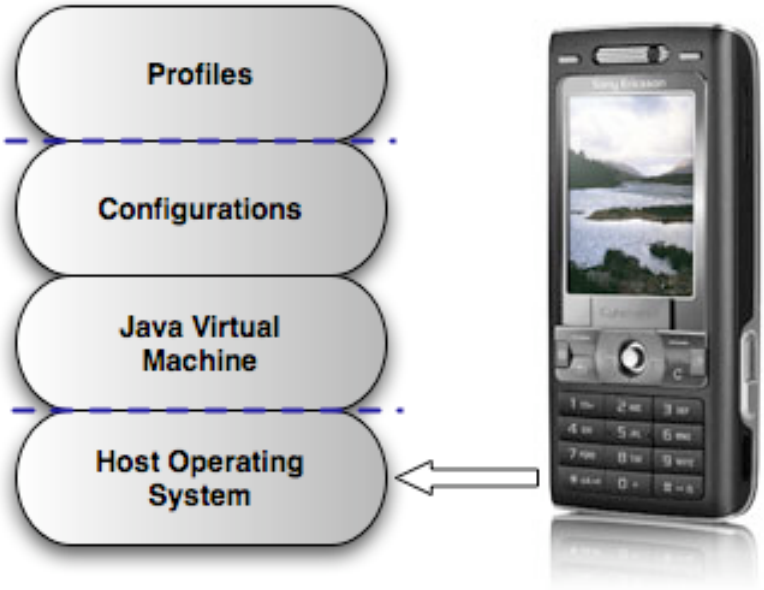

Figure 5.1: J2ME Architecture

Java Virtual Machine

The Java Virtual Machine layer is an implementation of a Java virtual machine that is customized for a particular devices host operating system and supports a particular J2ME configuration [25] [17].

The Java Virtual Machine (JVM) supporting small mobile devices with slow processors and limited memory is called the Kilobyte Virtual Machine (KVM). This virtual machine is in the range of 40 to 80 Kbytes - hence the name Kilobyte Virtual Machine. Devices targeted by the KVM have typically 16- or 32-bit processors and a minimum total memory of 128 kilobytes. To launch an application on top of KVM, a Java Application Manager (JAM) serves as an interface between the native operating system on the device and the KVM [25] [17].

Configurations

The Configuration layer defines the minimum set of Java Virtual Machine features and core Java class libraries available on a particular category of devices. This category of devices represent a particular market segment and can be thought of as the lowest common denominator of Java platform features that a developer can assume will be available on all devices [25] [17].

Currently, there exists two J2ME configurations; the Connected Limited Device Configuration (CLDC), and the Connected Device Configuration (CDC). CLDC is the smaller of the two configurations, designed for devices with slow processors and limited memory. This would typically be mobile phones, pagers and

PDAs [32]. Such devices usually have either 16- or 32-bit CPUs, and a minimum of 128 KB to 512 KB of memory available for the Java platform implementation and associated applications. The other configuration available, CDC, is a superset of CLDC and designed for devices that have more memory, faster processors, and greater network bandwidth, such as TV set-top boxes, in-vehicle telematics systems, and high-end PDAs. CDC includes a full-featured Java virtual machine, and a much larger subset of the J2SE platform than CLDC. Most CDC-targeted devices have 32-bit CPUs and a minimum of 2MB of memory available for the Java platform and associated applications [25] [17].

Profiles

The Profile layer defines the minimum set of Application Programming Interfaces (APIs) available on a particular group of devices. Profiles are implemented upon a particular configuration. The idea is that a category would include several different groups of devices. Devices that are members of the same category have fundamental features in common, while devices that are members of the same category and the same group offers equal functionality to the developer. When implementing applications in J2ME, applications are written for a particular profile and are thus portable to any device that supports that profile, i.e. devices that can be classified into the same category and group. A device can however support multiple profiles [25] [17].

The only profile currently developed for the CLDC configuration is the Mobile Information Device Profile (MIDP). It is designed for mobile phones and entry-level PDAs and offers core application functionality required by mobile applications. This includes API classes related to interface, persistence storage, networking, and application management. Together with the CLDC, MIDP provides a complete J2ME runtime environment. For the CDC configuration there are developed several profiles. The Foundation Profile, the Personal Profile and the Personal Basis Profile all adds different functionality for different types of devices supporting CDC [25] [17].

MIDlets

A Java application intended for a CLDC device is called a MIDlet [32], and must be formatted into a Java Archive (a JAR file) to run on the device. To enable distribution of third party MIDlets, developers must generate metadata files associated with each JAR file. This metadata files are called Java Application Descriptor files (JAD files) and contains information that the Java Application Manager (JAM) uses to verify and configure the MIDlet at loading time [17].

5.2 Bluetooth

This chapter gives a brief introduction to the Bluetooth technology. As the Peer2Me framework currently only uses Bluetooth as wireless network medium, a deeper understanding of how Bluetooth works will be useful to perform our redesign of the framework.

5.2.1 What is Bluetooth?

Bluetooth is a low cost, low power, short-range radio technology intended to replace cable connections between mobile phones, PDAs and other mobile devices. It can clean up your desk considerably, making wires between your workstation, mouse, laptop computer etc. obsolete. The idea that resulted in the Bluetooth technology was born in 1994 when the Swedish company Ericsson Mobile Communications decided to investigate the feasibility of a low-power, low-cost radio interface between mobile phones and their accessories. Ericsson soon realized that for the technology to succeed, there must be a critical mass of mobile devices using their new short-range radio technology, so in 1997 they decided to give the technology away for free. Only a year later, the five companies Ericsson, IBM, Intel, Nokia and Toshiba held simultaneous press conferences in England, USA and Japan announcing that the companies would join to develop a free, open specification for short-range wireless connectivity. The new specification was named ”Bluetooth”, and the five companies created a ”Bluetooth Special Interest Group” (SIG) that would be responsible for developing the new specification [28]. Today, the Bluetooth SIG promoter members include: Agere, Ericsson, IBM, Intel, Microsoft, Motorola, Nokia, and Toshiba. Promoter companies are highly engaged in the strategic and technical development of Bluetooth wireless technology [36].

5.2.2 Origin of the name

The Bluetooth technology is named after a tenth-century Danish Viking King, Harald Blåtand (english; Harald Bluetooth), who united and controlled Norway and Denmark. Blåtand was King of Denmark and Norway from 935 and 936 respectively, to 940, and contributed greatly to the unification of warring tribes from Denmark (including Skåne, present-day Sweden, where the Bluetooth technology was invented) and Norway. The name Bluetooth was chosen because of Harald’s ability to unite. The developers of Bluetooth hoped that the Bluetooth technology would unite the world as Harald Bluetooth united Norway and Denmark. Bluetooth likewise was intended to unify different technologies like computers and mobile phones. The Bluetooth logo merges the Nordic runes analogous to the modern Latin H for ”Harald” and B for ”Bluetooth”. The logo can be seen in Figure 5.2

Figure 5.2: The Bluetooth logo

5.2.3 Communicating via Radio Waves

A radio wave is a pulse of electromagnetic energy. Radio waves are generated when a transmitter oscillates at a specific frequency, and the faster it oscillates, the higher the frequency gets. To amplify and broadcast

the radio waves, an antenna is used, and to receive the radio signals a radio receiver is needed. Because different frequency ranges are used for different types om communications, the receiver must be tuned to a specific frequency.

Bluetooth operates in the license-free Industrial Scientific Medical (ISM) band at 2.4GHz [28]. This band is currently used by a wide range of devices such as:

* 2.4GHz cordless house telephones
* 802.11 wireless computer networks
* Baby guards/monitors
* Garage-door openers
* Some emergency radios
* Microwave ovens

In order to avoid interfering with other protocols using the 2.4GHz band, the Bluetooth protocol utilize a technique called "spread spectrum frequency hopping". This means that the Bluetooth radio signals hops among 79 frequencies between 2.402GHz and 2.480GHz (at 1MHz intervals), up to 1600 times per second.

Bluetooth devices are available in three different power classes [28]:

* Class 1 (100 mW): A range at up to 100 meters.
* Class 2 (2.5 mW): The most common used class. A range up to 10 meters.
* Class 3 (1 mW): A range up to a maximum of 1 meter. Rarely used.

5.2.4 Bluetooth Transfer Rate

In theory, implementations with the Bluetooth 1.0 Specification should be able to reach a maximum speed of 1 Mbps. This is however the gross data transfer rate (including the overhead), so the perceived rates are some lower. Bluetooth supports both symmetric and asymmetric transmission. At symmetric transmission (same speed in both directions), the maximum speed is 432.6Kbps. At asymmetric transmission (high speed in one direction, low speed in the other), the maximum speed are 721Kbps out and 56Kbps back. These speeds applies however just to data transmission. Voice signals are transferred with a maximum rate of 64Kbps in both directions [28].

The Bluetooth 2.0 Specification is backwards compatible with version 1.0. The main enhancement is the introduction of Enhanced Data Rate (EDR) of 2.1 Mbit/s. Technically devices supporting version 2.0 have a higher power consumption, but the three times faster rate reduces the transmission times, effectively reducing power consumption to half that of 1.0 devices [35].

5.2.5 Bluetooth Security

Bluetooth uses the SAFER+ [1] [33] algorithm for authentication and key generation. The E0 [2] [23] stream cipher is used for encrypting packets. This makes eavesdropping on Bluetooth-enabled devices more difficult.

5.2.6 Piconets and Scatternets

When two or more Bluetooth devices establish a connection all within the same signal range, we say that they have created a special type of personal area network (PAN) called a piconet (see Figure 5.3). Theoretically, a Bluetooth piconet can consist of up to a maximum of eight interconnected Bluetooth devices. While one device acts as a master node, the rest act as slaves [28].

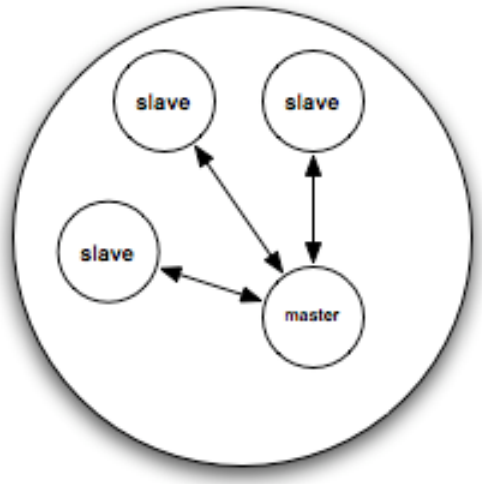

Figure 5.3: A piconet comprising four nodes

A device in one piconet can also communicate with another device in another piconet. This would interconnect the piconets into a scatternet shown in Figure 5.4. To establish communication between piconets in a scatternet, some nodes will have to get the responsibility for forwarding packets between piconets on behalf of other nodes. This would require advanced routing algorithms [28] [40].

[1] SAFER+ (Massey et al, 1998) was submitted as a candidate for the Advanced Encryption Standard and has a block size of 128 bits. The cipher was not selected as a finalist. SAFER+ was included in the Bluetooth standard as an algorithm for authentication and key generation.

[2] The E0 stream cipher generates a sequence of pseudorandom numbers and combines it with the data using the XOR operator. The key length may vary, but is generally 128 bits.

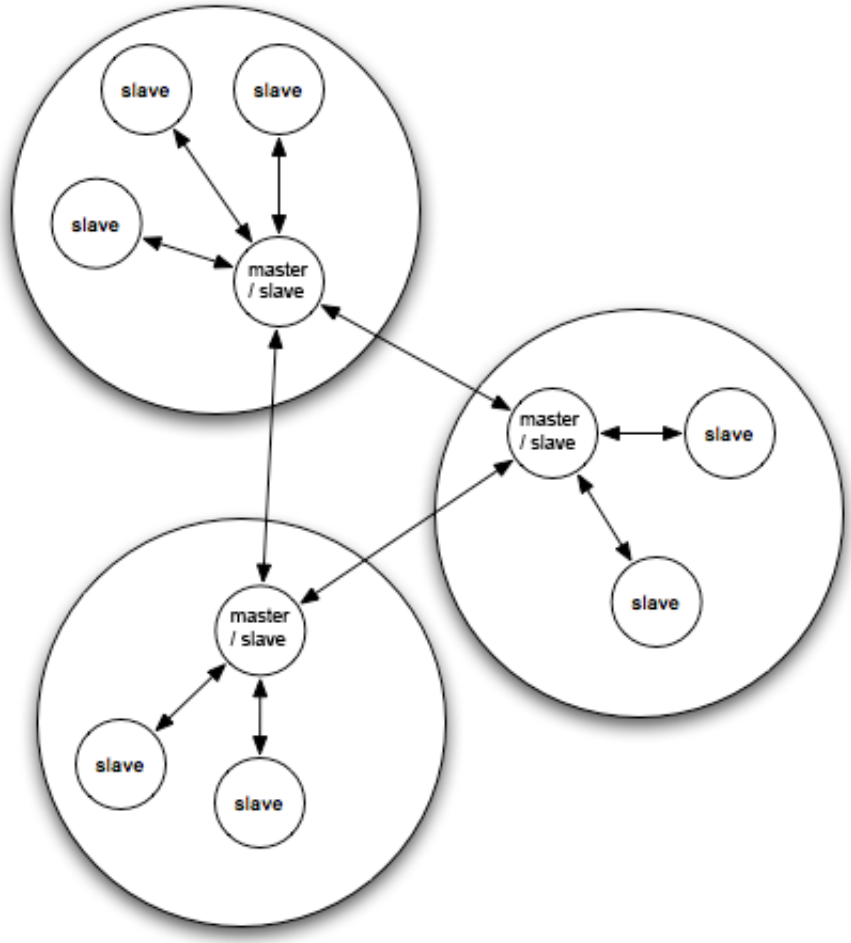

Figure 5.4: A scatternet comprising three piconets

CHAPTER 6

The Original Peer2Me Framework

Peer2Me is the name of a framework for developing mobile collaborative applications on mobile phones utilizing Personal Area Networks (PANs). The first version of the Peer2Me framework, which we will redesign and optimize in this master thesis, is the result of a depth study [30] and a master thesis [31] written by Lund and Norum during their studies at the Norwegian University of Science and Technology (NTNU). The framework is developed using J2ME technology [26] and currently supports Bluetooth as wireless network medium. Hereby we will refer to the original version of the Peer2Me framework as Peer2Me version 1.0 (v1.0), and the new improved version will be named Peer2Me version 2.0 (v2.0).

This chapter contains an overview of the requirements specified for Peer2Me v1.0. It also contains an explanation of essential domain concepts and a description of the Peer2Me v1.0 architecture. Later in this report, we will present a new set of requirements and a modified architecture, documenting Peer2Me v2.0. The original requirements, concepts and the design can be read in detail in the Peer2Me depth study from 2004 [30] and the Peer2Me master thesis from 2005 [31].

6.1 Peer2Me v1.0 Domain Concepts

Before we present Peer2Me v1.0 [31] with its properties and design, we will in this chapter explain some central concepts used in v1.0 of the framework. These concepts and the relations between them are essential for understanding the framework [21]:

Framework: "A framework is a set of classes that embodies an abstract design for solutions to a family of related problems."

Node: A node is a logical representation of a peer.

Network: The network module in the framework is an abstraction of the network layer. Applications built upon the Peer2Me framework will not use the network layer directly, but access it through the framework instance.

Service: A service is provided by an application and supported by zero or more nodes.

Group: A group is a collection of nodes providing the same service and communicating using a homogenous network. Every group must have a master node administering the group. Once the group has a master node, remaining nodes will act as slaves using the master node to communicate with each other.

Message: A message is the entity that can be exchanged between nodes connected in a group.

Application: An application is the software running on a mobile device using the Peer2Me framework.

6.2 Peer2Me v1.0 Functional Requirements

When eliciting the requirements for the Peer2Me framework, Lund and Norum chose the requirements composed by Sveen and Kirkhus in [38] as a basis. Sveen and Kirkhus had, however, set out to design and implement a full scale peer-to-peer framework for mobile collaboration. This included interconnection of several Bluetooth piconets requiring advanced routing protocols for sending messages using multiple hops to reach their destinations. To ensure that the framework could be realized, Lund and Norum chose to remove some of the requirements specified by Sveen and Kirkhus [38]. As a result of the work done by Lund and Norum, an implemented version of the Peer2Me framework is available, and according to their master thesis [31], the framework has functional requirements as listed in Table 8.1.

6.3 Peer2Me v1.0 Non-functional Requirements

The non-functional requirements for the Peer2Me framework are listed in Table 6.2.

6.4 Peer2Me v1.0 Design

This chapter contains a brief overview of the design of Peer2Me v1.0. A more detailed explanation of the design can be read in Lund and Norums master thesis [31]. In the following, we will present the Peer2Me v1.0 packages and explain the functionality in the framework.

When designing the Peer2Me framework, Lund and Norum chose to focus on developing a flexible framework highly independent of network technology. The intention was to reduce the work needed to migrate the framework to other network mediums such as Bluetooth, ZigBee, WLAN and others. To achieve this, a layered architectural model as viewed in Figure 6.1 was chosen.

The leftmost part of the figure shows the layers of the Peer2Me framework. Applications are strictly restricted to use an interface (layer) provided by core functionality offered from underlying layers. Further, this layer (named "Framework" in Figure 6.1) uses a generic network interface to control technology specific Network

Peer2Me v1.0 Functional Requirements	*Description*
FR 1	The framework supports mobile phones.
FR 2	The framework supports creation of ad hoc networks.
FR 3	The framework supports connection to an existing ad hoc network.
FR 4	Nodes in a network are able to exchange messages.
FR 5	The framework can create groups of nodes related to a specific application.
FR 6	The framework supports multicasting and broadcasting of messages within a group.
FR 7	The framework can search for other mobile phones supporting the same service.
FR 8	The framework supports creation of groups as closed or open.
FR 9	The framework allows a node to try to join a closed group.
FR 10	The framework allows users in a closed group to reject other nodes to join the group.
FR 11	The framework allows a node to join an open group.
FR 12	The framework is able to present decision messages to the user.
FR 13	The framework is able to present information to the user about framework related events.
FR 14	The framework supports different kinds of network mediums.
FR 15	The framework offers an interface which makes the applications independent of the underlying network medium that is in use within the system. Because of this interface, the applications that are using the system do not need to make any kinds of adjustments to fit a specific network implementation.
FR 16	The framework is able to identify where a transfer originated from. The purpose is to be able to send direct replies to a given device.
FR 17	The framework includes mechanisms for storing objects.
FR 18	The framework includes mechanisms for retrieving stored objects.

Table 6.1: Peer2Me v1.0 Functional Requirements

Non-Functional Requirements	*Description*
Non-FR 1	The framework is able to transfer messages fast enough for real time interaction. It is said to be 'fast enough' as long as normal length text messages give the impression of appearing instantly on the remote phones.
Non-FR 2	The framework is able to detect the disconnection of nodes within a group and notify relevant applications and nodes about this.
Non-FR 3	The framework adapts to errors that arise due to the unstable nature of wireless networks.
Non-FR 4	The framework prevents applications from getting access to messages not addressed to them.

Table 6.2: Peer2Me v1.0 Non-Functional Properties

modules. It is this module/layer that easily can be substituted and implemented with technologies such as Bluetooth, ZigBee or WLAN. The bottom layer is J2ME itself along with the specific network technology APIs. The rightmost part of the figure labeled ”Domain” contains the abstractions of the domain concepts Node, Group, Service and Message defined in Chapter 6.1.

Each layer in the architectural overview in Figure 6.1 is related to a specific package in the Peer2Me framework v1.0. A logical view of these packages is shown in Figure 6.2.

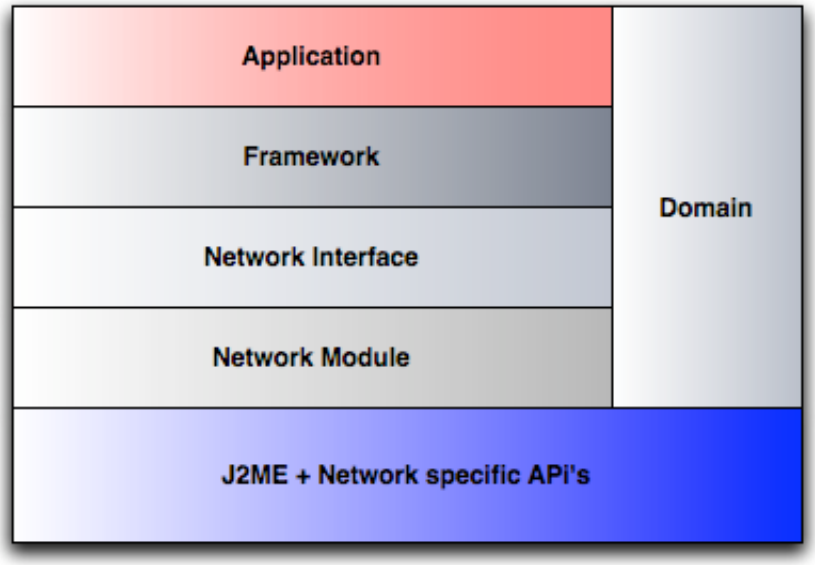

Figure 6.1: Architectural overview of the Peer2Me framework

The framework package contains the functionality offered to the applications using the Peer2Me framework v1.0. The domain package relates directly to the box labeled domain in the overview in Figure 6.1. The package named network contains the network interface that the framework uses to communicate with the network module. The Bluetooth package is a specific implementation of the network module layer, and can, as earlier mentioned, be easily replaced by network packages supporting other network technologies. In addition to the packages related to the layers in Figure 6.1, a package named util is added. This package contains different kinds of utilities and provides support functionality for the applications using the Peer2Me framework v1.0. For a throughout description of all classes in each of the packages, read Chapter 8.4 in Lund and Norums master thesis [31].

6.5 Known Problems

In our depth study from 2005 [39], we analyzed and evaluated Peer2Me framework v1.0 thoroughly and came up with quite a few suggestions for improvements. These were the main problems with Peer2Me v1.0:

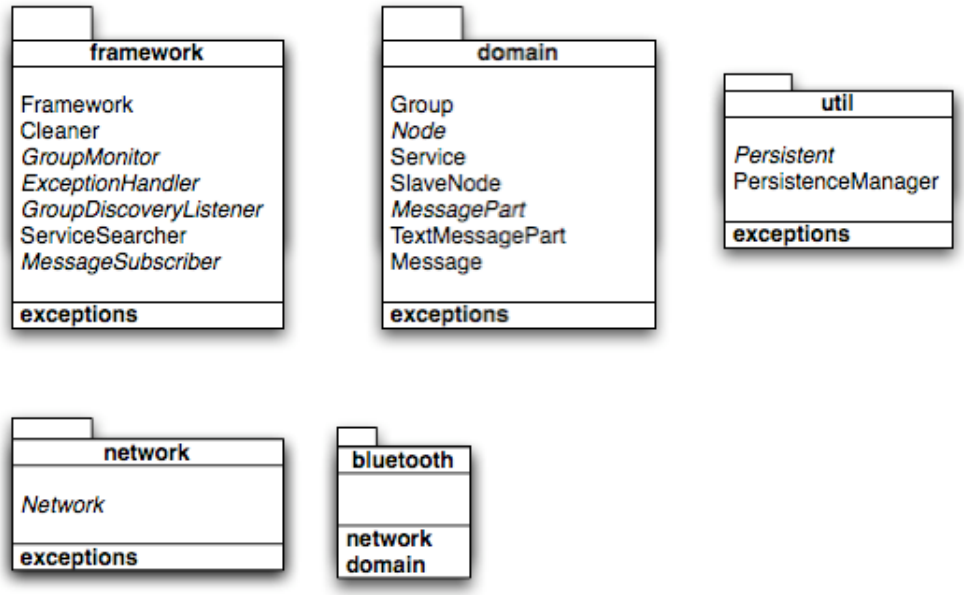

Figure 6.2: A logical view showing the main packages in the Peer2Me framework v1.0

* We found both the documentation and the commenting in the code to be inadequate and this caused us to spend too much time investigating how the different parts of the Peer2Me v1.0 work together and how to integrate an application with the framework.

* We found the architecture of Peer2Me v1.0 somewhat cumbersome and this caused us to spend the first couple of weeks trying to figure out how to develop our application. In order to use the framework, one must create a group, a service, set listeners, messagesubscribers, exceptionhandlers, monitors etc. Finding out how to do this was very time-consuming.

* We also discovered some bugs in Peer2Me v1.0. The search for other devices failes sometimes and the framework does not discover that other devices become disconneted. Sending a text containing \n from one node to another does not work. The transfer of the text is ended when \n occur.

* The framework uses the Debug MIDlet implemented along with the development of Peer2Me v1.0, therefore it cannot be compiled without this MIDlet.

* The bluetooth network layer is not entirely separated from the rest of the framework. According to Lund and Norum [31], the specific network layer can easily be replaced without affecting the framework itself. This is not the case at all, since many of the framework classes that don't belong to the network layer contains Bluetooth specific code. Even the MIDlets are required to choose to act as a master or a slave, which is specific for the Bluetooth protocol.

* Exceptions that occur in the framework are not thrown by the methods causing the errors. Instead, a single method is called every time an exception occur. This is a bad solution, because doing it this way prevents the MIDlet developers from taking action at the point where the exception occurred.

In addition to find smart solutions to the problems stated above, the Peer2Me framework v1.0 lacks important functionality like the possibility to send files of any kind over the network.

CHAPTER 7

Related Work

In this chapter we will look into projects that are similar to Peer2Me in technology and intention. The concept must include peer-to-peer topography, some sort of collaboration and be intended for use on mobile devices.

7.1 JXTA

JXTA is short for Juxtapose, that means side by side. It draws a parallel between peer-to-peer and client/server, web based computing - the two are juxtapose.

The JXTA project [24] was started by some researchers at Sun Microsystems. It's goal is to explore a vision of distributed network computing using peer-to-peer topology, and to develop basic building blocks and services that would enable innovative applications for peer groups. It is now a open source project under the Apache Software License and has the following three objectives:

* **Interoperability** - across different peer-to-peer systems and communities.
* **Platform independence** - multiple/diverse languages, systems, and networks.
* **Ubiquity** - every device with a digital heartbeat.

JXTA is defined to be independent of programming languages, so that it can be implemented in C/C++, Java, Perl, and numerous other languages. The Java binding, JXTA2SE, is the most mature of these. The protocol is specified as a set of XML messages. This means heterogeneous devices with completely different software stacks can interoperate with the JXTA protocols. The Peers in a JXTA based network can advertise and discover other resources, communicate with each other via "pipes" and cooperate dynamically to form peer groups.

Many of these features resembles those of the Peer2Me framework, but there are some differences:

* The Java binding of JXTA, JXTA Java SE, requires Java 2 Standard Edition (J2SE) to run. This makes it unsuited for most mobile devices on the market.

* There is no support for Bluetooth as a network medium, which also makes it unsuited for mobile devices since Bluetooth is the most common means of communication on such devices.

7.1.1 JXTA-J2ME (JXME)

To overcome this limitations a side project called JXTA-J2ME (JXME) is started [7]. The purpose of JXTA-J2ME is to provide a JXTA compatible functionalities on devices using the Connected Limited Device Configuration (CLDC) and the Mobile Information Device Profile 2.0 (MIDP), typically a mobile phone or a PDA.

The JXME was first designed as a proxy based peer-to-peer solution, relying on a central device acting as a proxy between the peers. This prevents "real" peer-to-peer operated, ad hoc networks. In the newest version however, this proxy is removed. The main disadvantage with both solutions is the lack of Bluetooth support.

7.1.2 Jadabs-CLDC

Jadabs-CLDC was developed during the semester work JXME-Bluetooth for a Mobile Phone (J2ME/CLDC) by René Müller at the Information and Communication System Group of the Swiss Federal Institute of Technology (ETH) Zurich [10]. Jadabs is a dynamic lightweight container for small devices. Combined with the JXME messaging system it can be used to build applications and service for a dynamical environment. The original Jadabs version could not run on a CLDC/MIDP based system, but Jadabs-CLDC is ported to cope with this limitation.

Jadabs-JXME-BT is a component for Jadabs-CLDC that implements a Bluetooth Transport Layer using the JSR-82 API for controlling the Bluetooth device. A peer in a JXME-BT initiated peer-to-peer network can operate in two different modes; normal peer and rendezvous peer. Normal peers can typically be a mobile phone, while more power devices like notebooks are used as rendezvous peers. A rendezvous peer can communicate via several interfaces simultaneously, e.g. Bluetooth and TCP/IP.

7.2 Ergon - J2ME Wireless Application Framework

The J2ME Wireless Application Framework is developed by Ergon Informatik AG, Zuurich. The framework, as described on their webpage [9], offers a complete set of client side and server side technologies, components and tools for building end to end Java based Wireless Business solutions and services. Some of the features include client and server components for encryption, authentication, user interface, data communication and server side data management. It supplements the standard CLDC and MIDP libraries with a set of

additional classes and development tools and supports open standards like SSL, HTTP and TCP/IP for communication.

7.3 BEDD

BEDD is a software package containing several applications supporting mobile collobaration and communication. BEDD is developed and maintained by the BEDD Corporation [8] and is currently implemented for the Symbian Series 60 OS. BEDD was first introduced in Singapore in May 2004, but is now introduced worldwide.

BEDD utilize Bluetooth as transport medium for message exchange between connected devices. There exists little information about how BEDD is actually implemented. This is due to the commercial nature of the software. Anyhow the structure is a typical framework with a central core module with separate applications implementing different types of functionality.

We have included a description of some of the applications below:

BEDDmates BEDDmates features a short profile describing yourself that is shared with all BEDD enabled phones in the proximity. A match analysis is performed and the user is alerted when the criteria of a match is met. When these "BEDDmates" are registered one can use all the other BEDD applications to communicate and interact with them.

BEDDbuddies Users that are added to the BEDDbuddies list will generate a notification when they come within range of you. They can then be contacted via BEDDtalk, BEDDchat, SMS, MMS, Call or E-mail.

BEDDshare Allows users to share software and files with other connected users. The application uses the recipients SMS inbox and built in Bluetooth to transfer the selected data.

7.4 JSR-259: Ad Hoc Networking API

The JSR-259 Ad Hoc Networking API [13] is a Java Community Process (JCP) started as a joint effort of several mobile phone vendors. These include BenQ/Siemens, Motorola, Panasonic and Nokia. In addition Sun Microsystems also participates in the development. The API will support communication between nodes in an ad hoc network implemented on mobile devices with J2ME support. The idea is that the API will enable developers to create peer-to-peer applications running on mobile devices.

Some of the features of the completed API is described in the JSR-259 Early Draft Review [13] as:

* Service Discovery
* Service Registration
* Service Availability Alert

* Service Capability Inquiry
* Remote Service Consumption

7.5 Conclusion

There are not many active projects in this research field and very few of them are similar enough to be compared with Peer2Me. The Jadabs-CLDC project stands put as the solution most similar in both technology and usage with Bluetooth as communication medium and J2ME implementation platform. Our challenge will be to create a improved user experience and enhance the Usability for the developers using the Peer2ME framework in comparison with the Jadabs solution.

Part III

Redesigning the Peer2Me Framework v1.0

CHAPTER 8

Requirements

In this chapter we will describe the requirements we have elicited for this project. The requirements are divided into three types; functional, non-functional and environmental.

8.1 Functional Requirements

As the intent of this project is to redesign and optimize the Peer2Me framework, rather than creating a new framework from scratch, we will try to use as many of the functional requirements described in Lund and Norum's master thesis [31] as possible. This is because we want to keep most of the framework's properties and concentrate on improving the usability for developers. The functional requirements specified in Lund and Norum's master thesis [31] are shown in Table 8.1.

In Chapter 7.1 of Lund- and Norum's master thesis [31] the Peer2Me framework's Functional Requirements are presented in Table 7.1. Since we are using an adapted UP development process, we will present these requirements in the form of Use Cases in the following. If necessary we will remove and add requirements to best suite our vision for the redesigned framework.

In our depth study [39] we gave a short description of the Use Case concept and we will now repeat this before we present the Use Cases of this project.

Use Cases are used to capture and document the potential requirements of the system [1]. A typical Use Case Modeling process proceeds as this:

* Find the system boundary - What is *part* of the system and what is *external* to the system. The boundary is drawn as a box in the diagram, with the actors outside the box and the Use Cases inside the box.

Functional Requirements	*Description*
FR 1	The framework must support mobile phones.
FR 2	The framework must support creation of ad hoc networks.
FR 3	The framework must be able to connect to an existing ad hoc network.
FR 4	Nodes in a network must be able to exchange messages.
FR 5	The framework must be able to create a group of nodes related to a specific application.
FR 6	The framework must support multicasting and broadcasting of messages within a group.
FR 7	The framework must be able to search for other phones supporting the same service.
FR 8	The framework must support the creation of groups as closed or open.
FR 9	The framework must support to allow a node to try to join a closed group.
FR 10	The framework must allow users in a closed group to reject other nodes to join the group.
FR 11	The framework must support to allow a node to join an open group.
FR 12	The framework must be able to present decision messages to the user.
FR 13	The framework must be able to present information to the user about framework related events.
FR 14	The framework must be able to support different kinds of network mediums.
FR 15	The applications must be independent of what network medium that is currently in use within the system. The application that is using the system to handle network traffic should not have to know what kind of network medium that is used by the device or make any kinds of adjustment to fit a specific network implementation.
FR 16	The framework must be able to identify where a transfer originated from. To be able to send direct replies to a given device, it must be possible to see where a transfer originated from.
FR 17	The framework must include a mechanism for storing objects.
FR 18	The framework must include a mechanism for retrieving stored objects.

Table 8.1: Peer2Me Functional Requirements

* Find the actors - An actor specifies a role that some external entity adopts when interacting with the system directly. It may be either a user or another system.

* Find the Use Cases:

    - Specify the Use Case - After creating a Use Case diagram and identifying the actors and main Use Cases, each of the Use Cases must be specified. We have chosen to create only one top level Use Case diagram for the system, namely Use Case 0 (see Figure 8.1). Then we specify the Use Cases found in Chapter 8.1.1.

    - Create scenarios - For complex Use Cases one can choose to make scenarios based on the Use Case. A scenario represents one specific path through a Use Case. We will not produce any scenarios for any of our Use Cases.

* In addition a Use Case model contains several relationships between actors and the Use Cases.

Identifying actors To identify the actors we have to consider who or what are going to use the system, and what roles they have in connection to the system. Some of the questions one can ask that helps identifying actors:

* Who or what uses the system?
* What roles do they play?
* Who installs the system?
* Who maintains the system?

Actors are always external to the system and they interact directly with the system. Furthermore they represent roles related to the system and not actual persons or objects, but one person or thing may play many roles simultaneously. The actors of a Use Case ought to have a short and descriptive name that clearly states the actors' role or function.

Use Case specification There are no UML standard for specifying Use Cases, but the template we will use for our Use Cases (e.g. Use Case 1 in Table 8.2) is in common use. In addition to a name and a description the Use Cases consist of several elements:

* Actors - A list of the actors involved.
* Preconditions - What must be true before the Use Case can start? Constraints on the state of the system
* Flow of events - The steps of the Use Case under normal circumstances.
* Alternative flow of events - If something goes wrong in the normal flow of events this flow come into use.
* Postconditions - These conditions must be true at the end of the Use Case.

The top level Use Case diagram for the Peer2Me framework is illustrated in Figure 8.1.

8.1.1 Use Cases

In this chapter all the Use Cases representing the functional requirements of the Peer2Me framework are presented using the template described above.

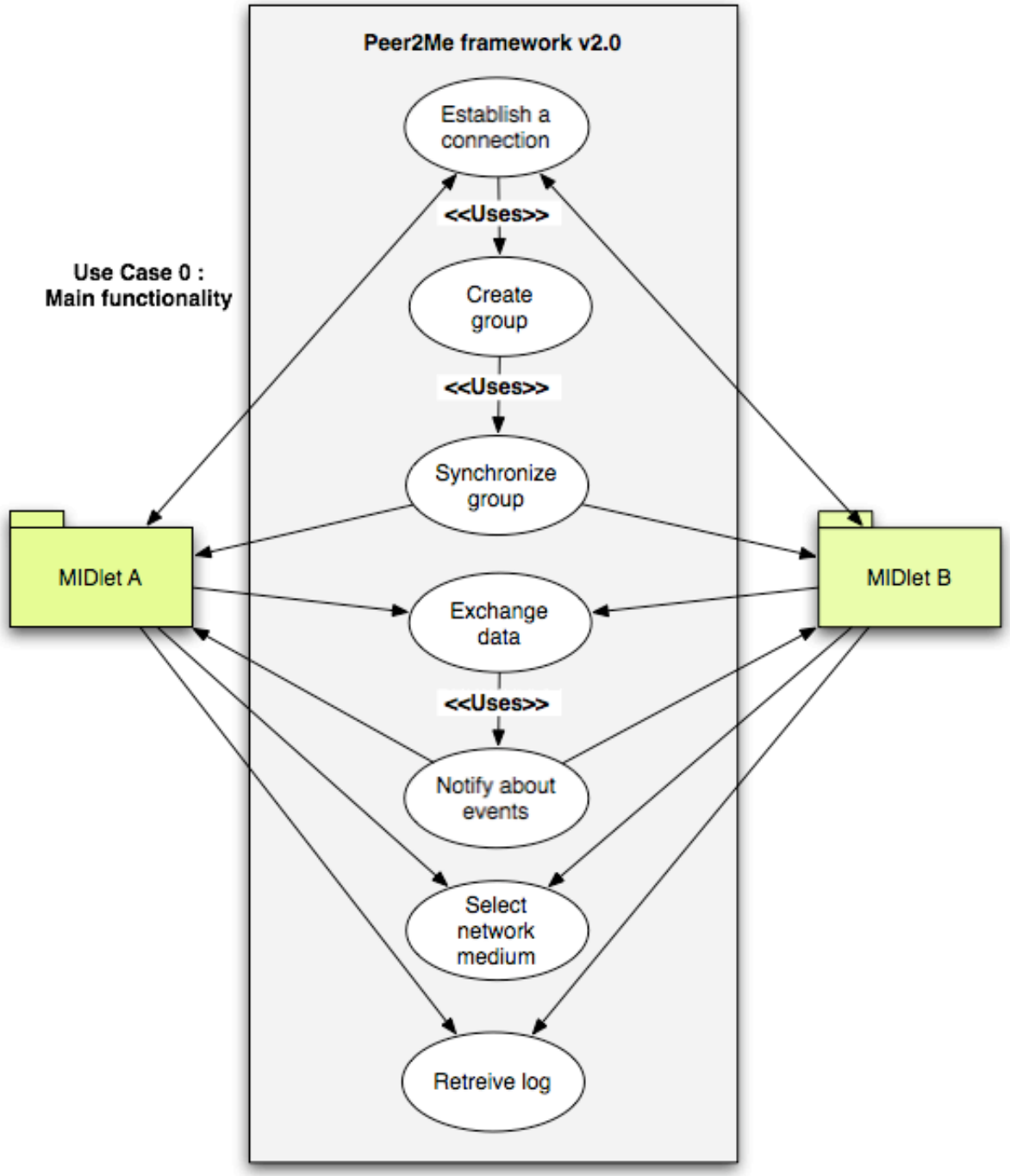

Figure 8.1: Use Case 0 model

Use Case : Establish a connection
ID: UC1
Actors: ∗ MIDlet A...n
Preconditions 1. The Peer2Me framework has to be installed on a mobile device with network support.
Flow of events 1. The MIDlet starts. 2. The MIDlet makes an ”initiate connection” call on the Peer2Me framework. 3. The Peer2Me framework initiates a search for other devices. 4. A list of discovered devices running the Peer2Me framework is presented to the MIDlet. 5. One or several devices are selected, and the MIDlet asks the Peer2Me framework to connect to the selected device(s).
Postconditions 1. The ad-hoc network is established.
Alternative flow 1 1. The MIDlet starts. 2. The MIDlet makes an ”initiate connection” call on the Peer2Me framework. 3. The Peer2Me framework initiates a search for other devices. 4. No devices are found.
Postconditions 1. No ad-hoc network is established.

Table 8.2: Use Case 1

Use Case : Create group
ID: UC2
Actors: * The Peer2Me framework
Preconditions 1. The Peer2Me framework has to be initiated on a mobile device with network support.
Flow of events 1. The MIDlet makes an "initiate" call on the Peer2Me framework. 2. The framework creates a group and adds a representation of the local device.
Postconditions 1. A group is created.

Table 8.3: Use Case 2

Use Case : Synchronize groups
ID: UC3
Actors: * The Peer2Me framework * MIDlet A...n
Preconditions 1. The Peer2Me framework has to be initiated on a mobile device with network support. 2. A connection between two or more devices must be established (8.2). 3. A group with two or more participants have been created (8.3).
Flow of events 1. The framework sends a synchronize message to all the participants in the group. 2. Each participant receives the synchronize message and updates the local representation of the group.
Postconditions 1. All participants have updated their local representation of the group.

Table 8.4: Use Case 3

Use Case : Exchange data
ID: UC4
Actors: * MIDlet A...n
Preconditions 1. The Peer2Me framework has to be initiated on a mobile device with network support. 2. A connection between two or more devices must be established (8.2). 3. A group with two or more participants have been created (8.3). 4. The groups must be synchronized (8.4).
Flow of events 1. MIDlet A wants to send some data via the framework. 2. The framework wraps the data into a data package and sends it over the connection to the recipient(s). 3. MIDlet n receives the data via an event notification (8.6).
Postconditions 1. Some data is exchanged.
Alternative flow 1 1. MIDlet A wants to send some data via the framework. 2. The framework wraps the data into a data package and sends it over the connection to the recipient(s). 3. The data could not be sent over the connection.
Postconditions 1. No data has been exchanged.

Table 8.5: Use Case 4

Use Case : Notify about events
ID: UC5
Actors: * The Peer2Me framework * MIDlet A
Preconditions 1. The Peer2Me framework has to be initiated on a mobile device with network support. 2. A connection between two or more devices must be established (8.2). 3. A group with two or more participants have been created (8.3). 4. The groups must be synchronized (8.4)
Flow of events 1. An event occurs in the framework. 2. The MIDlet is notified about the event.
Postconditions 1. The MIDlet is notified about the framework event.

Table 8.6: Use Case 5

Use Case : Select network medium
ID: UC6
Actors: * MIDlet A
Preconditions 1. The Peer2Me framework has to be installed on a mobile device with network support. 2. The Peer2Me framework must support the desirable network medium.
Flow of events 1. The network medium is specified by the MIDlet. 2. The MIDlet initiates the framework with the specified network medium.
Postconditions 1. The framework is initiated with the specified network medium.

Table 8.7: Use Case 6

Use Case : Retrieve log
ID: UC7
Actors: * MIDlet A
Preconditions 1. The Peer2Me framework has to be initiated on a mobile device with network support.
Flow of events 1. The MIDlet retrieves a log from the framework.
Postconditions 1. A log is available to the MIDlet.

Table 8.8: Use Case 7

The following functional requirements described in Lund- and Norums master thesis [31] are not converted into Use Cases in the above; 1, 7, 8, 9, 10, 11, 13, 15, 16 and 18. They are either removed or merged into a Use Case covering another functional requirement.

8.2 Non-functional Requirements

Non-functional requirements are requirements of a slightly more diffuse character than the functional requirements described in the Use Cases above. They are often not so clearly stated by the users and stakeholders of a system, but are nonetheless very important for the user satisfaction and the architecture [1]. These requirements are unsuited for Use Case representation and in Lund- and Norums master thesis [31] the Non-functional requirements are presented in a table. The content of this table is presented in Table 6.2. We will adapt these requirements, refine them and finally present them as ways to achieve these quality attributes; Usability, Performance, Modifiability, Availability, Security and Testability. To easily understand our choices of non-functional requirements, the definitions of the different quality attributes also found in Chapter 4 is repeated.

Non − Functional Requirements	*Description*
NFR 1	The framework must be able to transfer messages fast enough for real time interaction. By fast enough, we mean that normal length text messages should give the impression of appear- ing instantly on the remote phones.
NFR 2	The framework must be able to detect the disconnection of nodes within a group and notify relevant applications and nodes about this
NFR 3	The framework must adapt to errors that arise due to the unstable nature of wireless net- works.
NFR 4	The framework must prevent applications from getting access to messages not addressed to them.

Table 8.9: Peer2Me v1.0 Non-Functional Requirements

8.2.1 Usability

The definition of Usability:

> ”Usability is concerned with how easy it is for the user to accomplish a desired task and the kind of user support the system provides.”

In the problem definition found in Chapter 1.2 we have stated that we will improve the architecture and simplify the interface of the Peer2Me framework. Both of these statements are connected to the Usability quality attribute. An improved architecture and interface will increase the developer’s ability to make full use of the framework. These are the factors to achieve this quality:

* Simple and intuitive interface between the framework and the applications.
* Well documented and commented code.
* Descriptive naming conventions for methods, variables and objects.

8.2.2 Performance

Performance is described as:

> ”Performance is about timing. Events (interrupts, messages, requests from users, or the passage of time) occur, and the system must respond to them.”

Applications built upon the Peer2Me framework are intended to run on different types of mobile devices, performance is a considerably factor. Many of these devices have limited resources, like memory and CPU power, and this can affect the performance of the developed applications in a negative way. The first non-functional requirement found in Table 6.2 concerns this quality attribute:

* The framework must be able to support real time interaction. This means that data packages must be transferred with a rate of at least 10kB/s (NFR 1 made more precise).
* The framework must support pure peer-to-peer communication. By excluding a centralized routing node a potential bottleneck is avoided.

8.2.3 Modifiability

The definition of Modifiability:

> ”Modifiability is about the cost of change. It brings up two concerns; What can change (the artifact)? - When is the change made and who makes it (the environment).”

The Peer2Me framework needs a large degree of modifiability for several reasons. The consept of a independent network layer demands a simple method of adding new network modules. This process must be as simple as possible and have little impact on the parts of the framework not related to the network. The framework itself is a work in progress and will be subject of further change in the future. To simplify the work of those who are to continue our work we will take precautions to make the Peer2Me framework as intuitive and modifiable as possible. It is important that:

* The network layer is completely independent from the rest of the framework and that there is a interface between them that ensures the possibility to replace the network module in the future.
* We use known patterns and best practices in our architecture to simplify future modifications. To further increase this quality the characteristics of documentation and code described in the Usability section are useful (see Chapter 8.2.1).

8.2.4 Availability

This is said about Availability:

> ”All approaches to maintaining availability involve some type of redundancy, some type of health monitoring to detect a failure, and some type of recovery when a failure is detected.”

The second and third non-functional requirements of the Peer2Me framework found in Table 6.2 focus on Availability. We wish to continue these two requirements into our project, and also add another requirement making the network less vulnerable.

* The framework must be able to detect the disconnection of nodes within a group and notify applications and nodes about this (NFR 2). The possibility to detect whether or not the other nodes in the ad hoc network is present is crucial for the applications running the Peer2Me framework. This creates awareness between the users.

* The framework must adapt to errors that arise due to the unstable nature of wireless networks (NFR 3).

* The framework must support pure peer-to-peer communication. By excluding a centralized routing node, the network is less vulnerable to errors.

8.2.5 Security

Tactics for achieving security can be divided into three different categories; resisting attacks, detecting attacks and recovering from attacks. In *Software Architecture in Practice* [5] we find this analogy:

> "Putting a lock on your door is a form of resisting an attack, having a motion sensor inside of your house is a form of detecting an attack, and having insurance is a form of recovering from an attack."

The last of the non-functional requirements found in Table 6.2 is a Security requirement:

* The framework must prevent applications from getting access to messages not addressed to them (NFR 4).

To ensure the integrity and confidentiality of the messages sent between nodes in an ad hoc network it is important that a message reaches only the designated receiver(s). These two mechanisms contribute to the achievement of this goal:

* Authenticate - The nodes unique network address can be used for this purpose.

* Authorize - The Peer2Me framework has to support the concept of closed groups were authorization is done by a password.

8.2.6 Testability

The goal of Testability as a quality attribute is:

> "Allow easier testing when an increment of software development is completed."

A well known tactic to achieve Testability is ”Record/Playback”. This includes using a log to catch information in runtime and display this information for testing purposes. Testing can become useful and necessary in several scenarios:

* Testing to locate flaws in the design during redesign of the Peer2Me framework.
* Good testability is important to ease further development of the framework in the future.
* Developers using the finished Peer2Me framework will need to do testing when creating applications based upon the framework.

8.3 Environmental Requirements

In this section will describe in short the environment that is needed by the application to execute properly.

J2ME - MIDP 2.0 Both the Peer2Me framework and Peer2Me applications are implemented in Java and therefore depend on J2ME [26] and MIDP 2.0 support.

Operative System (OS) As long as the mobile device that is to run the application implements the technologies mentioned in the previous point, the Operative System is irrelevant. This is achieved with the platform independence of the Java programming language.

Memory The size of the Peer2Me framework, and with it the need for memory, will vary with the number of included applications in the JAR archive.

Display The applications GUI should be operational and functional on any type of display on any mobile device that fulfills the previous requirements of this list.

Bluetooth The mobile device must have support for Bluetooth [16] as this is the only network medium currently implemented in the Peer2Me framework. In addition the J2ME API JSR-82 must also be implemented to allow the access to Bluetooth functionality by a J2ME application running on the mobile device.

CHAPTER 9

Design

This chapter will focus on the design and architecture of the redesigned Peer2Me framework. We will begin describing the high level architecture and the architectural patterns we have incorporated into our design. Subsequently a more detailed description of the different classes of the framework is provides.

The Design phase is part of the Analysis and Design task of RUP's iterative development loop (see Figure 2.1), and we have based our design on the requirements elicited from the Use Cases as well as the non-functional- and environmental requirements (see Chapters 8.1, 8.2 and 8.3).

9.1 High Level Architecture

The architecture of Peer2Me framework v2.0 is strictly module based with a seperate package for each major type of classes. The top level structure of the architecture can be seen in Figure 9.1. This model represents a Module-Decomposition view where the modules are related to each other by the "is a sub module of" relation [5], e.g. all the packages are sub modules of the entire system and the bluetoothNetwork package is a sub module of network.

One of the main features of the architecture is that all the Bluetooth specific code are located in a sub-package of the network package which in turn means that the rest of the code are completely network independent. In that way a future conversion of the framework to an alternative network technology will not affect the "non network" classes of the framework. It will be possible to incorporate a new network module directly into the current system without any changes. This has been one of the main goals of this redesign from the start (see Chapter 1.2).

We will now describe shortly the main concepts of the different packages of the Peer2Me v2.0 framework.

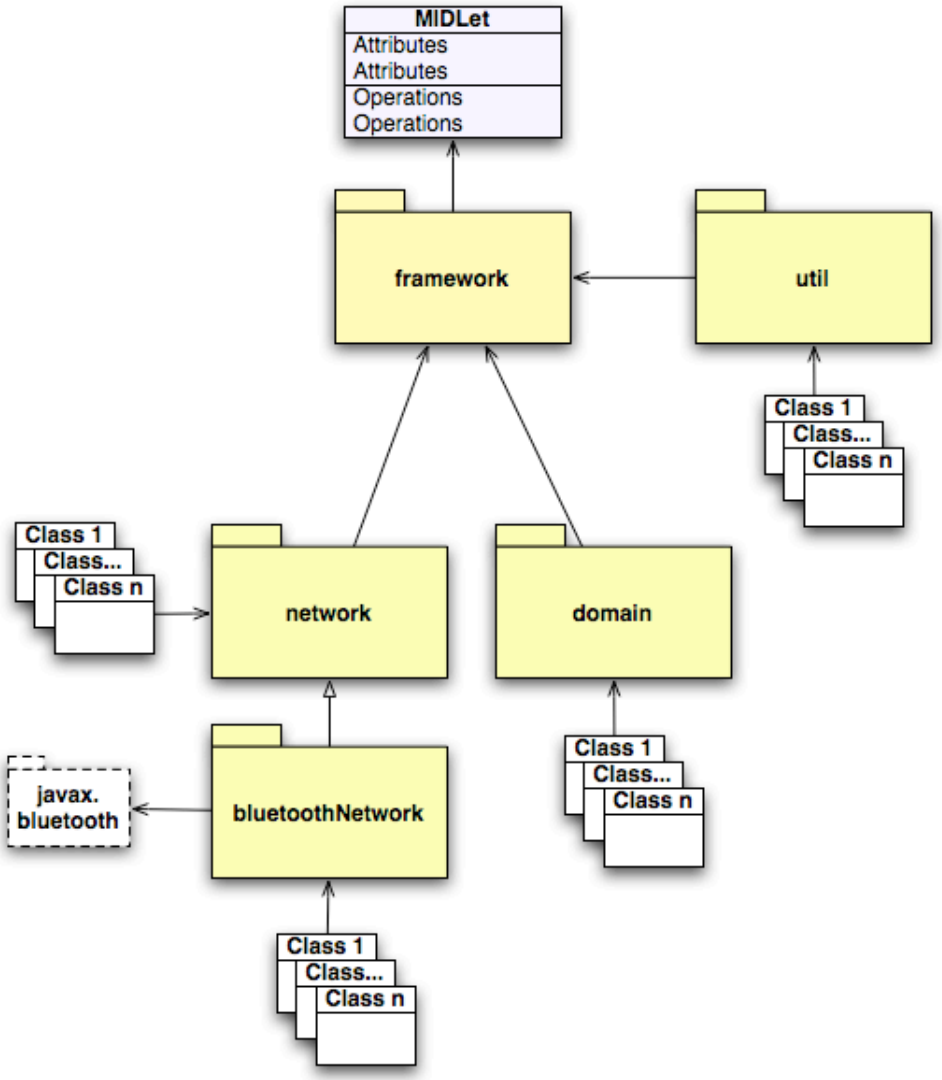

Figure 9.1: Module Decomposition View

Framework This package contains the FrameworkFrontEnd class which is the core of the system. All communication between the user interface of the MIDlet and the functionality of the framework are done through this class. Two interfaces connects the MIDlet with the FrameworkFrontEnd. Due to this design the rest of the framework is hidden for the developers using it.

Domain This package consists of the classes based on the conceptual domain of the framework that represents "real time" objects. Group, Node and DataPackage are some of the classes of the domain package.

Network The network package holds the classes concerning all network communication. The Connection-Listener listens for incoming connections from other devices and the NodeConnection class represent

a connection to each of the remote nodes whitin the same group.

BluetoothNetwork This sub package of the network package is, as the name reveals, responsible for all network operations that are specific to the Bluetooth technology that the Peer2Me framework is currently using. By replacing this package with another network module, the framework could communicate over e.g. a WLAN network.

Util The classes of the Util package contains helpful functionality used by the other classes of the framework. The classes of the Util package can be used from any other class in any package of the Peer2Me.

Midlets This is the actual MIDlets running upon the framework. They contain the functionality and user interface that is unique for every application. The MIDlets can take advantage of all of Peer2Me's functionality through the framework interface.

9.2 Detailed description

Figure 9.2 is a high level illustration of all the classes in the framework and the relations between them, while Figure 9.3 shows the full class diagram of the Peer2Me framework v2.0. The Log class is deliberately left out in Figure 9.3 because it has reference to almost all the other classes and interfere with the empirical quality of the diagram.

In this chapter we will briefly discuss the function of each of the classes and how they interact.

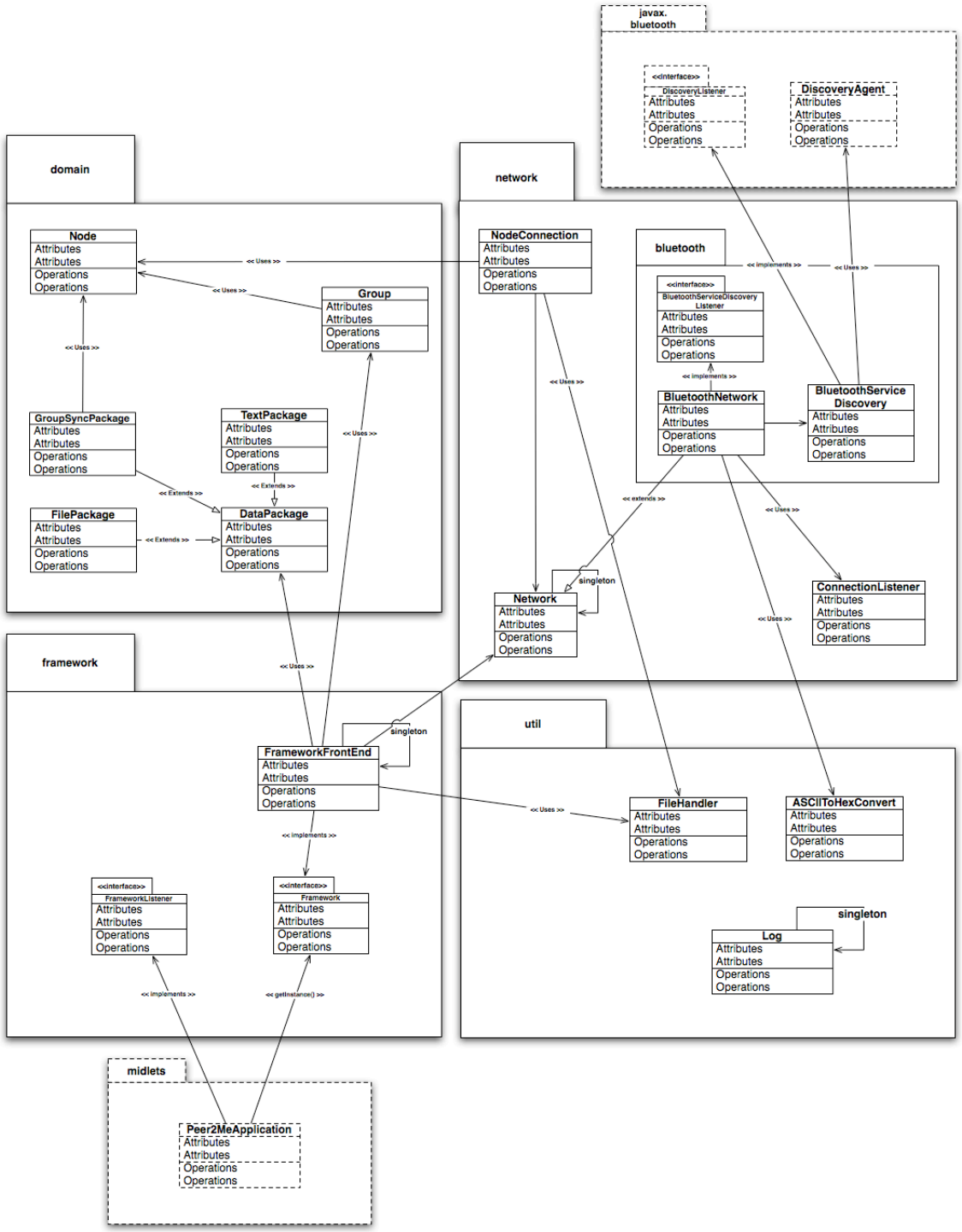

Figure 9.2: Peer2Me Architecture

Figure 9.3: Class Diagram

9.2.1 Framework package

This package contains the core of the Peer2Me framework v2.0 and the interfaces used by the MIDlets.

FrameworkFrontEnd This is the main class of the Peer2Me framework. It manages and connects the resources and functions of the framework. It also handles all communication and interaction with the individual MIDlets running the framework.

Framework $\ll interface \gg$ The Framework interface acts as a "facade" for the entire Peer2Me framework as the methods in this interface is the only methods the MIDlets running the framework needs access to. The MIDlets receive a reference to the FrameworkFrontEnd class casted into this interface when the getInstance() method is called to get an instance of the framework. See Chapter 9.3 for more information about the facade pattern.

FrameworkListener $\ll interface \gg$ The FrameworkListener interface must be implemented by all MIDlets running the framework. It ensures that the Framework can access a set of methods in the MIDlet in order to notify the MIDlet about various events.

9.2.2 Domain package

The classes of this package represents the conceptual domain objects that affects the framework. Each class is based on a "real" object found in the domain, and contains the properties of that object.

Group Holds the information related to the Group object. The connected nodes in the ad hoc network are participants in the Group. Through methods participants can be added and removed, and a list of all the participants can be retrieved.

Node The Node class represents a node in the ad hoc network, i.e., a mobile device running the framework. Contains information about the name of the Node and network address.

DataPackage This class is the super class of the different type of packages that can be sent between nodes in the network. The address of the sender and the recipients of the DataPackage is stored in the super class, along with the type of package. There are currently three types of packages and they are described below.

GroupSyncPackage This type of Datapackage is a package used internally in the framework to synchronize the groups containing the participants. The participant performing the groupsync uses its own group as content of the package. All the receivers synchronizes their groups based on the information found in the GroupSyncPackage.

FilePackage A FilePackage is sent between two or more participants as a part of a file transfer. The package contains the file path and length of the file to transfer, so that the receiver can handle the incoming stream of data and transform it back into a copy of the file.

TextPackage To send text between participants one can use the TextPackage. It contains some length of text that the recipient(s) can retreive and present to the user of the MIDlet.

9.2.3 Network package

The Network package is made up of the networking classes of the Peer2Me framework. A sub package of this package is created for each new network technology that is implemented. For now the BluetoothNetwork package is the only sub package.

Network This is the super class of the technology specific network classes. Methods that are equal for all the sub classes are located in this super class, and there are abstact methods that the sub classes have to implement. The getInstance() method of the Network class returns a reference to the preferred network sub class.

NodeConnection Each Node has a NodeConnection that holds the data streams to and from the remote device that the Node object represents. These data streams are used to transfer Datapackages between the devices.

ConnectionListener When a MIDlet initiates the Peer2Me framework a ConnectionListener is started. Its task is to listen for incoming connection attempts from other devices running the same MIDlet built upon the framework. When a incoming connection is detected, a Node representation is created representing the connecting device.

The following classes are part of the *bluetoothNetwork* package which is a subpackage of *network*.

BluetoothNetwork This class is a Bluetooth specific sub class of the Network class and implements all the abstract methods of it's parent class in a Bluetooth context. It uses the Bluetooth Java API[1], JSR-82, to perform operations on the Bluetooth hardware of the mobile device.

BluetoothServiceDiscovery To discover, identify and connect to other bluetooth enabled devices the BluetoothNetwork class uses methods located here. A DiscoveryAgent is called to perform the discovery process and the result is returned through method calls to this class.

BluetoothServiceDiscoveryListener $\ll interface \gg$ To ensure that the BluetoothServiceDiscovery class can return it's results to the BluetoothNetwork class, the BluetoothNetwork class have to implement the BluetoothServiceDiscoveryListener interface.

9.2.4 Util package

All classes that have some kind of a "helper" function is located in the Util package. All the classes of the framework can use them.

Log The Log contains four different kinds of logs, an exception log, a connection log, a data package log and a debug log. They can be used to log events from anywhere in the framework, and the logs can be retrieved later to get information about the execution of the MIDlet. This is particularly useful on mobile devices as they do not have a console to display runtime and debug information.

FileHandler The FileHandler is used to read, write and create files on the local file system of the device.

[1] Application Programming interface

ASCIIToHexConverter This small class converts a ASCII string into a hexadecimal number. It is used by the BluetoothNetwork class to create a unique UUID[2].

9.3 Design Patterns

This section contains a description of the design patterns used in our architecture. A pattern is a general solution to a common problem in software design and is useful to create high quality code. It is not a finished design though, but a kind of template for solving that problem as it occurs in different situations and applications. Typically the pattern outlines the relationships and interactions between objects on a high level. No finished classes or implementations are specified [12].

Singleton Pattern The singleton pattern is used in object oriented programming to avoid more than one instantiation of a class. This is necessary when exactly one object is needed to coordinate actions across the system and when the number of objects should be limited due to efficiency. The pattern is implemented by creating a class with a method that creates a new instance of the class if one does not exists. To force other classes to use this method of instantiation the constructor of the class is made private or protected. In cases with multithreading the singleton pattern is vulnerable because the instantiation method could be called simultaneously by two threads. This is often solved by making the getInstance() method synchronized, thereby introducing mutual exclusion.

We have taken advantage of this pattern in our Log-, FrameworkFrontEnd- and Network classes to be certain that there exists one and only one instance of each of the classes. This is important to ensure that all method calls is performed on the intended

Facade Pattern This pattern is based on the idea that a facade provides a simplified interface to a large portion of code. This is useful to make e.g. a large library of classes easier to use, since the facade provides a set of simple methods that allows the user to perform common operations. It also reduces dependencies between the code on opposite sides of the facade, that in turn makes it more straight forward to make alterations to the library classes.

The Framework interface of the Peer2Me framework is designed using this pattern. A MIDlet running the framework has access only to the methods of this interface and it acts as a facade hiding the rest of the framework.

Observer Pattern All J2Me GUI classes make use of a CommandListener to listen for command actions invoked by the user. These actions are performed when a ”soft button” is pressed in a MIDlet. The listener detects what ”soft button” is pressed and starts an operation based on this.

[2]Universally Unique Identifier

CHAPTER 10

Implementation

In this chapter we will describe the implementation method used during implementation of the Peer2Me framework v2.0. To illustrate the main concepts of our design and architecture we have included a selection of code sniplets from out implementation. The code samples will present a typical flow of events within the framework when a certain operation is performed and there will be thorough explanations of each example.

10.1 Implementation Method

In our depth study [39] we used the concept of Pair Programming with great success and have chosen to use this as our main implementation method in this project as well. We will now repeat the definition of and introduction to Pair Programming.

This definition of Pair Programming is found at pairprogramming.com[1].

> Two programmers working side-by-side, collaborating on the same design, algorithm, code or test. One programmer, the driver, has control of the keyboard/mouse and actively implements the program. The other programmer, the observer, continuously observes the work of the driver to identify tactical (syntactic, spelling, etc.) defects and also thinks strategically about the direction of the work. On demand, the two programmers can brainstorm any challenging problem. Because the two programmers periodically switch roles, they work together as equals to develop software.

According to *All I really need to know about pair programming I learned in kindergarten* [42], pair programming is a powerful technique for productively code high quality software. By working together in pairs, the developers tackle the complexities of software development and the continuously inspections of

[1] www.pairprogramming.com

each others code ensures early and efficient removal of errors. In addition, they keep each other focused on the task and how to solve it in the most effective manner. Surveys show that programmers working in pairs are more confident in their solutions than those who work alone [42].

The article lists a number of "rules" that leads to successful pair programming. Some of the most important of these rules are:

Share everything *Two* programmers jointly produces *one* product and it is important that they see themselves as team and that they feel equally responsible for the result.

Play fair Although it is a good chance the two programmers have different levels of competence it is very important that they take turns as driver and observer respectively.

Drive your partner forward When one of the two are losing concentration it is easier to get encouraged and keep going when you work in pairs.

Stay positive Negative thoughts easily transfers to your partner.

Clean up Remove errors when they are detected.

Review independent work If one of some reason have to produce code independently it is efficient to review each others work. This way many defects are identified on an early stage.

Take breaks Working in pairs like this, demands a lot of concentration and it is important to take breaks to clear the mind every now and then.

We will try to follow these rules when it is time for us to start programming and this will hopefully give us the opportunity to code more and reduce error searching.

10.2 Implementation tools

The implementation of all Java source code along with deployment of the Peer2Me framework v2.0 was performed using the Eclipse IDE[2] in combination with the SUN Java Wireless Toolkit[3] and the EclipseME[4] Eclipse plugin. To be able to emulate file access we had to use a emulator found in the Sony Ericsson SDK 2.2.3 for the Java ME Platform[5]. Our choice of tools is based on experiences gained in earlier projects.

As we used both Windows and Mac OSX as platforms for development during this project, the Eclipse IDE was a obvious choice as implementation environment. Eclipse is Java based and with that it supports multiple operative systems. The SUN Java Wireless Toolkit however had to be modified slightly to run under Mac OSX as SUN does not offer a toolkit native to OSX. Using the SUN Java Wireless Toolkit for Linux combined with the preverifier from mpowerplayer[6] gave us the same programming capabilities on both

[2] www.eclipse.org
[3] http://java.sun.com/products/sjwtoolkit/index.html
[4] www.elipseme.org
[5] developer.sonyericsson.com
[6] http://mpowerplayer.com/for_developers.php

platforms. The only drawback is that the emulator functionality on Mac OSX is reduced and do not support Bluetooth emulations.

To keep track of all our work and all the changes in the source code we took advantage of the features offered by the Concurrent Versions System (CVS)[7]. It gave us the possibility to collaborate and work on the same files simultaneously regardless of our location. It also keep track of changes and updates our local copies so no data is lost and we could be certain that the code we are working with are the right one. Eclipse features a CVS client that makes updating and committing an entire project possible through the push of a single button.

[7] http://www.nongnu.org/cvs/

10.3 Source Code Examples

To present some of the main features of our architecture we will highlight some code sniplets and explain how the different parts interact. The examples are of code that represents and illustrates the architectural solutions and important aspects of our design. The code represents a typical flow of events from the initiation of the framework by the MIDlet, via the discovery process and synchronization of groups, to the sending and receiving of a text package. The whole process is described in the sequence diagram found in Figure 10.1. The "bubbles" containing numbers like 10.3.x refers to which sub chapter contains details about this specific method. We strongly recommend that this Sequence Diagram is read in parallel with the code examples to increase the understanding. In addition, the Peer2Me v2.0 Javadoc (Appendix C) can with advantage be used as a reference while reading the code.

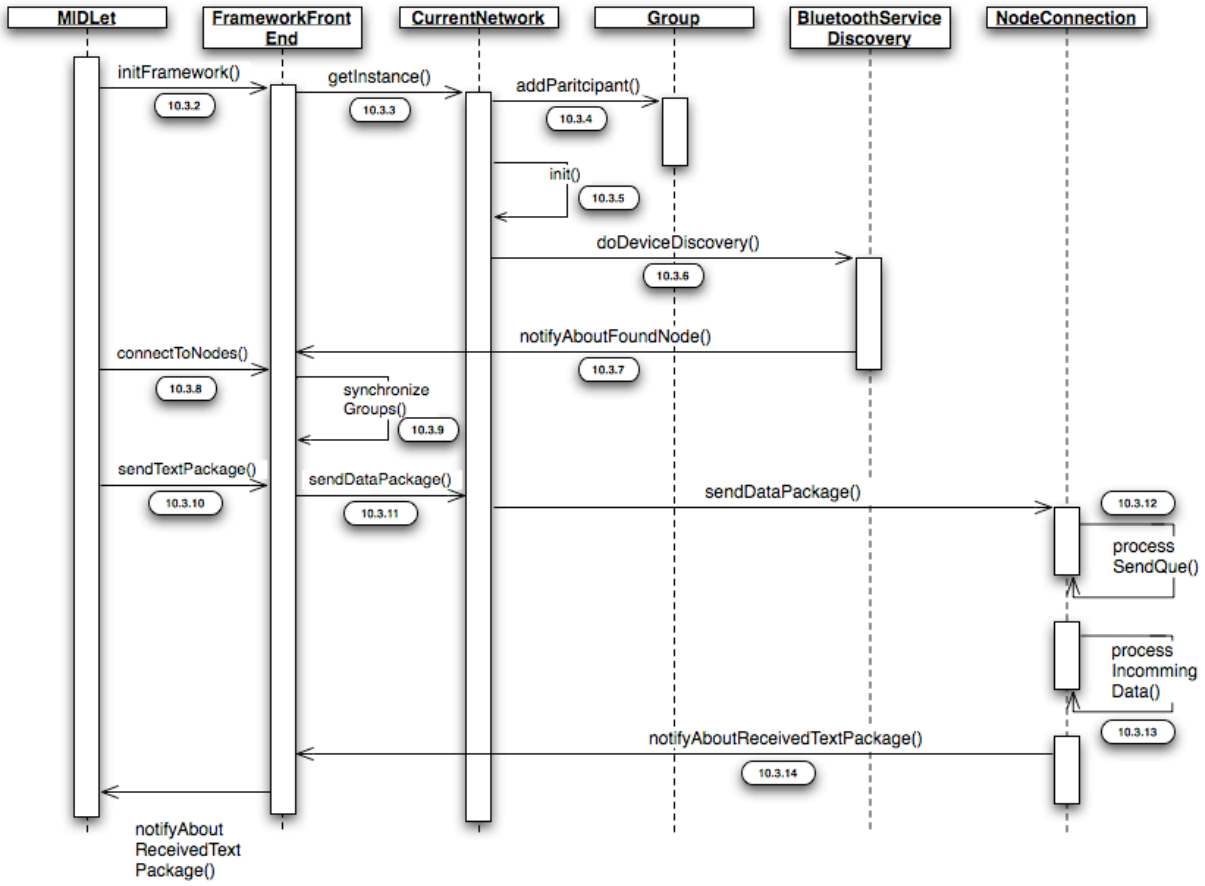

Figure 10.1: Sequence Diagram

10.3.1 The Framework interface

This first listing illustrates the Framework interface between the MIDlet and the rest of the Peer2Me framework. The methods of this interface is use by the MIDlet to make use of all the features of the framework. When the FrameworkFrontEnd.getInstance() method is called, a reference of type Framework is returned. When a MIDlet is run, the initFramework() method has to be called to initiate the framework before anything else can be done. The initFramework() method is described in the next chapter (10.3.2).

Listing 10.1: The Framework interface

```
package peer2me.framework;

import java.io.IOException;
import java.util.Enumeration;

/**
 *
 * This interface acts as a "facade" for the entire Peer2Me framework as the
 * methods in this interface is the only methods the MIDlets running the
 * framework needs access to. To use the Peer2Me framework, the MIDlets should
 * run the FrameworkFrontEnd.getInstance() which returns a
 * reference of type Framework. All framework services is then available
 * through this reference.
 *
 * @author Torbjørn Vatn & Steinar A. Hestnes
 */
public interface Framework{

  /**
   *
   * This method initiates the framework, and is the first method that should
   * be run after getting a instance of the framework. It initiates the
   * fundamental services offered by the framework.
   *
   * @param nodeName The name of the user of the MIDlet.
   * @param midletName The name of the MIDlet, eventually translated into a ServiceID
     * used to find other devices running the same MIDlet.
   * @param preferredNetwork Deciding which network implementation to use.
   *
   * @throws ClassNotFoundException The input preferredNetwork is invalid
   * @throws IllegalAccessException The input preferredNetwork is invalid
   * @throws InstantiationException The input preferredNetwork is invalid
   * @throws IOException Error initiating framework
   * @throws Exception Error initiating framework
   */
  public void initFramework(String nodeName, String midletName, String preferredNetwork) throws ClassNotFoundException,
```

```
                                        IllegalAccessException, InstantiationException, IOException, Exception;


    /**
     *
     * This method shuts down the framework and closes all the open network connections and streams.
     * It should be called before closing the MIDlet to clean up the network connections.
     *
     */
    public void shutdownFramework();


    /**
     *
     * This method starts a search for devices running the same MIDlet.
     * When such a device is found, the notifyAboutFoundNode() method
     * specified by the FrameworkListener interface is called.
     *
     * @throws IOException Thrown if the search crashes
     */
    public void startNodeSearch() throws IOException;


      /**
       *
       * This method connects multiple devices in a network.
       * When a connection is established, the notifyAboutParticipants()
       * method specified by the FrameworkListener interface is called.
       *
       * @param addresses The addresses of the devices to connect to.
       */
      public void connectToNodes(String[] addresses);


      /**
       *
       * This method sends a text package over the network. When the package
       * terminates to the recipients, they are alerted by the
       * notifyAboutReceivedTextPackage() method specified by the
       * FrameworkListener interface.
       *
       * @param recipients A list containing the addresses of the recipient nodes
       * @param textMessage The text message to be sent
       *
       */
```

```
82      public void sendTextPackage(String[] recipients, String textMessage);
83
84
85      /**
86       *
87       * This method sends a file package over the network. When the package
88       * terminates to the recipients, they are alerted by the
89       * notifyAboutReceivedFilePackage() method specified by the
90       * FrameworkListener interface.
91       *
92       * @param recipients A list containing the addresses of the recipient nodes
93       * @param filePath The path of the file to be sent
94       *
95       */
96      public void sendFilePackage(String[] recipients, String filePath);
97
98
99      /**
100      *
101      * This method returns a list of the files in the given root directory on the device
102      *
103      * @param root The path to the root directory
104      * @return An enumeration containing the names of the files in the root directory
105      */
106     public Enumeration getFileList(String root);
107
108
109
110 }
```

10.3.2 The initFramework() method in FrameworkFrontEnd.java

This method is vital to the execution of a MIDlet because the framework must be properly initiated before any calls can be made to it. The initFramework() method does a number of operations:

1. Creates a ”currentNetwork” instance using the getInstance() method in Network. The getInstance() method is described in the next chapter (10.3.3).

2. Sets the MIDlet name on the ”currentNetwork”. The name of the MIDlet is used to find other devices running the same MIDlet, hence to be able to connect in a network.

3. Creates a new Group and adds itself as a participant using the addParticipant() method in class Group. This method is listed in Chapter 10.3.4.

4. Calls the init() method on the ”currentNetwork” to initiate the network layer. The init() method is listed in Chapter 10.3.5.

Listing 10.2: The initFramework() method in FrameworkFrontEnd.java

```
/**
 *
 * This method initiates the framework, and is the first method that should
 * be run after getting a instance of the framework. It initiates the
 * fundamental services offered by the framework.
 *
 * @param nodeName The name of the user of the MIDlet.
 * @param midletName The name of the MIDlet, eventually translated into a ServiceID used to find other devices
   * running the same MIDlet.
 * @param preferredNetwork Deciding which network implementation to use.
 *
 * @throws ClassNotFoundException The input preferredNetwork is invalid
 * @throws IllegalAccessException The input preferredNetwork is invalid
 * @throws InstantiationException The input preferredNetwork is invalid
 * @throws IOException Error initiating framework
 * @throws Exception Error initiating framework
 */
public void initFramework(String nodeName, String midletName, String preferredNetwork) throws ClassNotFoundException,
                                    IllegalAccessException, InstantiationException, IOException, Exception{

  // Creates a Network instance
  currentNetwork = Network.getInstance(preferredNetwork);
    // Sets a reference to this class to be used in the Network class
    currentNetwork.setFrameworkFrontEnd(this);
    // Sets the applicationId to be used by the Network class
    currentNetwork.setApplicationId(midletName);
    // Creates a group that will be filled with nodes running the same application
```

```
107         group = new Group();
108           // Adds a representation of this (the local) node to the group.
109           localNode = new Node(nodeName,currentNetwork.getNodeAddress("localnode"));
110           group.addParticipant(localNode);
111           // Initiates the currentNetwork
112           currentNetwork.init();
113           // Creates the foundNodes Hashtable
114           foundNodes = new Hashtable();
115     }
```

10.3.3 The getInstance() method in Network.java

This method returns a reference to a subclass of Network, based on the preferredNetwork input parameter. In the current implementation the only network module available is BluetoothNetwork.

Listing 10.3: The getInstance() method in Network.java

```
/**
 *
 * This method returns an instance of the preferred network.
 * It is called from FrameworkFrontEnd.initFramework().
 *
 * @param preferredNetwork Indicating which network implementation to use.
 * @throws ClassNotFoundException The input preferredNetwork is invalid
 * @throws IllegalAccessException The input preferredNetwork is invalid
 * @throws InstantiationException The input preferredNetwork is invalid
 * @return The Network instance
 */
public static synchronized Network getInstance(String preferredNetwork) throws ClassNotFoundException,
                                                                    IllegalAccessException, InstantiationException{

  // A log instance
  log = Log.getInstance();

  if(singleton != null){
    return singleton;
  }else{
    try{
      // Fetching a instance of the preferred network class
      singleton = (Network)Class.forName(preferredNetwork).newInstance();

    }catch(ClassNotFoundException cnfe){
      log.logException("Network.getInstance()",cnfe, false);
      throw cnfe;
    }catch(IllegalAccessException iae){
      log.logException("Network.getInstance()",iae, false);
      throw iae;
    }catch(InstantiationException ie){
      log.logException("Network.getInstance()",ie, false);
      throw ie;
    }

    // Returning the singleton instance
    return singleton;
  }
}
```

10.3.4 The addParticipant() method in Group.java

This method adds a Node as a participant to the Group. If the Node is not a participant already it is simply added, while existing participants are updated with the new data. This is used during a synchronization of the groups. The Node objects are stored in a HashTable with the network address as the key.

Listing 10.4: The addParticipant() method in Group.java

```
    /**
     *
     * This method adds a node to the group as a participant.
     *
     * @param node The node to add as a participant.
     */
    public void addParticipant(Node node){
        // Adds the node only if it is not added already.

        // This test is necessary during groupsync
        if (!participatingNodes.containsKey(node.getAddress())){
          participatingNodes.put(node.getAddress(),node);
        }else{
            // If the node already exists in the participant list , this is the node that
            // initially discovered this node and was saved only with address and connection
            // Name is still missing and we have to add it
            if (node.getNodeName() != null){
              ((Node)participatingNodes.get(node.getAddress())).setNodeName(node.getNodeName());
            }

            if (node.getNodeConnection() != null){
                if (node.getNodeConnection().getConnection() != null){
                    // Important to start the connection!
                    ((Node)participatingNodes.get(node.getAddress())).startNodeConnection();
                  ((Node)participatingNodes.get(node.getAddress())).getNodeConnection().setConnection(
                                                        node.getNodeConnection().getConnection());
                }
            }
        }
    }
```

10.3.5 The init() method in BluetoothNetwork.java

When the Network.getInstance() method is called the init() method of the new "currentNetwork" is also called. It initiates the network module through several steps:

1. Creates a connectionURL to use in the connection process. The URL consists of some Bluetooth specific variables and a UUID[8] that indicates what MIDlet is running on this device.
2. Sets the local Bluetooth device discoverable, so it can be discovered by other devices searching.
3. Creates a new BluetoothServiceDiscovery instance that can perform a device discovery. The doDeviceDiscovery() method is listed in Chapter 10.3.6.
4. Creates and sets a new ConnectionListener that is listening for incoming connections from other devices performing a device discovery.

Listing 10.5: The init() method in BluetoothNetwork.java

```
/**
 * Initiates the network instance.
 * It is called from the FrameworkFrontEnd.initFramework()
 *
 * @throws BluetoothStateException Failed to initiate the network
 */
public void init() throws BluetoothStateException{

  isConnected = false;
  serviceSearchCompleted = false;
  serviceSearchFailed = false;


  // Sets the connectionURL used by the ConnectionListener
      String localNodeName = getFrameworkFrontEnd().getLocalNode().getNodeName();
      connectionURL = "btspp://localhost:"+getUUIDString()+";authenticate=false;encrypt=false;name="+localNodeName;

  // Have to set the local device discoverable
      try {
          LocalDevice.getLocalDevice().setDiscoverable(javax.bluetooth.DiscoveryAgent.GIAC);
      } catch (BluetoothStateException bse) {
          log.logException("ConnectionListener.ConnectionListener()",bse,false);
          throw bse;
      }

      foundNodes = new Hashtable();
      // Creates the class that contains low level Bluetooth discovery operations.
```

[8] Universally Unique Identifier

```
96          bluetoothServiceDiscovery = new BluetoothServiceDiscovery();
97
98      /* The ConnectionListener instance that listens for incoming requests from
99       * other nodes in discovery mode. When this node is discovered the "discoverer"
100      * can choose to create a connection between the two, and the remote node is
101      * represented by a node object localy on this node.
102      */
103         setConnectionListener(new ConnectionListener(connectionURL));
104  }
```

10.3.6 The doDeviceDiscovery() method in BluetoothServiceDiscovery.java

This method starts a discovery process using the DiscoveryAgent of the local Bluetooth device. This is a regular search for Bluetooth devices in the proximity and takes between 20 - 30 seconds to complete. Other methods of the BluetoothServiceDiscovery class (deviceDiscovered() and serviceDiscovered()) are called by the DiscoveryAgent whenever a device running the same MIDlet is discovered.

Listing 10.6: The doDeviceDiscovery() method in BluetoothServiceDiscovery.java

```
/**
 *
 * This method starts the discovery process.
 * It is called from BluetoothNetwork.searchForNodes().
 *
 * @throws BluetoothStateException Error getting reference to LocalDevice
 */
  public void doDeviceDiscovery() throws BluetoothStateException{

    uuids[0] = new UUID(uuidString, false);
    servicesFound = new Vector();
    devicesFound = new Vector();

      try{
          localDevice = LocalDevice.getLocalDevice();
      }catch(BluetoothStateException bse) {
          log.logException("BluetoothServiceDiscovery.doDeviceDiscovery()",bse, false);
          throw bse;
      }

      //Fetches the discovery agent of the local device
      agent = localDevice.getDiscoveryAgent();

      try {
        // The discovery agent starts the inquiry for other devices
          agent.startInquiry(DiscoveryAgent.GIAC,this);
      }
      catch(BluetoothStateException bse) {
        log.logException("BluetoothServiceDiscovery.doDeviceDiscovery()",bse, false);
      throw bse;
      }
  }
```

10.3.7 The notifyAboutFoundNode() method in FrameworkFrontEnd.java

When the DiscoveryAgent is finished searching for other devices the serviceSearchCompleted() method of the BluetoothServiceDiscovery class is called. Then the notifyAboutFoundNode() in class FrameworkFrontEnd is used to notify the framework of every found device. The found Nodes are stored in the foundNodes HashTable with their network addresses as keys, and if two or more Nodes has identical names, the names are modified by adding a number at the end. In this way every Node gets a unique name. The MIDlet is also notified about the found Node.

Listing 10.7: The notifyAboutFoundNode() method in FrameworkFrontEnd.java

```
/**
 *
 * This method is called from the nodeFound() method in the Network class whenever a node is found
 *
 * @param address The network address of the node
   * @param remoteNodeName The name of the found remote node
 */
public void notifyAboutFoundNode(String address, String remoteNodeName){

  // Here we add a number after equal node names to make them unique
    // We do this so we can set the node names as keys and the node addresses as values
    // The reason for doing this is that the node names will be displayed in the midlet
    // and after selecting a node name, the address should be sent to the framework.
    if (foundNodes.contains(remoteNodeName) || remoteNodeName.equals(localNode.getNodeName())){
        for (int i=-1;i<foundNodes.size();i++){
          if (!foundNodes.contains(remoteNodeName+" "+(i+2))){
            remoteNodeName = remoteNodeName+" "+(i+2);
            i = foundNodes.size();
          }
        }
    }

  // Stores the address and the name of the node in the foundnodes table
  foundNodes.put(address, remoteNodeName);
  midlet.notifyAboutFoundNode(address,remoteNodeName);
}
```

10.3.8 The connectToNodes() method in FrameworkFrontEnd.java

When the user of the MIDlet has chosen the Nodes he/she wants to connect to, this method in class FrameworkFrontEnd is called. It simply adds the selected Nodes to the Group and uses the syncronizeGroup() method to broadcast the Group to all the participants of the Group. This way all the selected Nodes gets connected and the groups on every node gets synchronized in one operation. Now, every Node in the Group can send data packages to all the other participants.

Listing 10.8: The connectToNodes() method in FrameworkFrontEnd.java

```
    /**
    *
    * This method establishes a connection to the chosen nodes.
    * After updating the local group, it synchronizes the groups on
    * all other participating nodes.
    * The method should be called from the MIDlet.
    *
    * @param addresses The addresses to the nodes to connect to.
    */
   public void connectToNodes(String[] addresses){
     // Creates Node objects based on the Vectors nodeNames and nodeAddresses
     for(int i=0; i<addresses.length; i++){
       getGroup().addParticipant(new Node((String)foundNodes.get(addresses[i]),addresses[i]));
     }
     // Synchronizes the groups on all connected nodes
     synchronizeGroups();
   }
```

10.3.9 The synchronizeGroups() method in FrameworkFrontEnd.java

When the user of the MIDlet has chosen which Nodes to connect to through the connectToNodes() method, this method is called to synchronize the Groups of all the participating Nodes. It sends a special GroupSyncPackage to all the participants found in the local Group, containing all the other Nodes. This way all participants have the means to synchronize their own Group.

Listing 10.9: The synchronizeGroups() method in FrameworkFrontEnd.java

```
176     /**
177      *
178      * This method is used to make the Framework syncronize the Groups on all the
179      * connected nodes. The result of running this method is that the method
180      * notifyAboutParticipants() is called on the MIDlet.
181      * It is called from the methods connectToNodes() and
182      * notifyAboutLostNode() in this class.
183      *
184      */
185     private synchronized void synchronizeGroups(){
186
187         // Creates a string table with the recipient addresses
188         Hashtable participatingNodes = group.getParticipatingNodes();
189
190         String [] recipients = new String[0];
191         // Only do this if there is more than this node in the group
192         if (participatingNodes.size()>1){
193         recipients = new String[participatingNodes.size()-1];
194           // Need a list of nodes to run a groupsync
195           Node[] nodes = new Node[participatingNodes.size()];
196           // Adds the local Node to the nodes[]
197           nodes[0] = localNode;
198           // Removes the local Node from the participatingNodes[]
199           participatingNodes.remove(localNode.getAddress());
200
201           Enumeration addresses = participatingNodes.keys();
202           int counter = 0;
203
204           while(addresses.hasMoreElements()){
205                 String address = (String)addresses.nextElement();
206                 // Does not add the local node
207                 recipients [counter] = address;
208                 // Fetches the Node objects from participatingNodes
209                 nodes[counter+1] = (Node)participatingNodes.get(address);
210                 counter++;
211           }
212
```

```
213            // Sends a networkpackage to all participants to synchronize the group on all nodes
214            if(recipients.length!=0){
215                currentNetwork.sendDataPackage(new GroupSyncPackage(localNode,recipients,nodes),recipients);
216            }
217            // Adds the local Node to the group again
218            participatingNodes.put(localNode.getAddress(),localNode);
219        }
220
221        // Notifies the MIDlet about the participants of the group
222        notifyAboutParticipants();
223
224        // Logs the sending of the data package
225        String recipientNames = "";
226        for(int i=0; i<recipients.length; i++){
227            if(group.getNode(recipients[i])!=null){
228                recipientNames +="- "+group.getNode(recipients[i]).getNodeName()+" ("+recipients[i]+") \n";
229            }
230        }
231
232        if(recipients.length>0)log.logDataPackage("Sent a group sync package to:\n "+recipientNames);
233    }
```

10.3.10 The sendTextPackage() method in FrameworkFrontEnd.java

When all of the connecting and synchronizing are finished the framework are ready to perform other tasks on behalf of the MIDlet. Sending a text package could be such a task. By calling this method through the Framework interface, and applying the message to send, the MIDlet can send text to other MIDlets running on other Nodes. As long the text and a list of receivers is provided by the MIDlet, the Peer2Me framework performs the nessesary tasks to send the text package.

Listing 10.10: The sendTextPackage() method in FrameworkFrontEnd.java

```
    /**
    *
    * This method is used by the MIDlet to send a text package over the network.
    * When the package terminates to the recipients, the
    * notifyAboutReceivedTextPackage() method in this class is run.
    *
    * @param recipients A list containing the addresses of the recipient nodes
    * @param textMessage The text to be sent
    *
    */
   public void sendTextPackage(String[] recipients, String textMessage){

     // Logs the sending of the text package
       String recipientNames = "";
       for(int i=0; i<recipients.length; i++){
         recipientNames +="- "+group.getNode(recipients[i]).getNodeName()+" ("+recipients[i]+") \n";
       }
       log.logDataPackage("Sending textpackage to:\n"+recipientNames);

       TextPackage textPackage = new TextPackage(localNode,recipients,textMessage);
       // Passes the task of sending the data package over to the network
       if(recipients.length!=0)currentNetwork.sendDataPackage(textPackage, recipients);
   }
```

10.3.11 The sendDataPackage() method in BluetoothNetwork.java

When the sendTextPackage() method or the sendFilePackage() in the FrameworkFrontEnd class is called, they in turn call on this method to have the network layer perform the actual sending of the data package. It reconnects to each of the receiving Nodes to establish a data stream and places the data package in an outgoing que using the sendDataPackage() method of the NodeConnection.

Listing 10.11: The sendDataPackage() method in BluetoothNetwork.java

```
297     /**
298      *
299      * This method is used by the FrameworkFrontEnd to send a data package of
300      * any sort to a remote node.
301      *
302      * @param dataPackage The data package to be sent
303      * @param recipients A list containing addresses to the recipient nodes
304      *
305      */
306     public void sendDataPackage(DataPackage dataPackage, String[] recipients){
307
308       // A Vector containing the addresses to the nodes that could not be reached
309       Vector addressesToLostNodes = new Vector();
310
311       for(int i=0; i<recipients.length; i++){
312         // If the node has been removed/disconnected in the meantime
313         if(getFrameworkFrontEnd().getGroup().getNode(recipients[i])==null){
314           // do nothing
315         }else{
316
317           // Connects to the remote node if the connection never has been opened or if it has been closed
318           NodeConnection nodeConnection = getFrameworkFrontEnd().getGroup().getNode(recipients[i]).getNodeConnection();
319           if(nodeConnection!=null){
320             if(nodeConnection.getConnection()==null){
321               // Establishes a connection to the recipient
322               // This method waits until the new connection is ready (or not)
323             connectToNode(recipients[i]);
324             }else if(nodeConnection.getSendQueueSize() == 0){
325               // If the que is empty, the connection has been closed, and we need a new one
326               nodeConnection.setConnection(null);
327               // Establishes a connection to the recipient
328               // This method waits until the new connection is ready (or not)
329             connectToNode(recipients[i]);
330             }
331           }else{
332           // Establishes a connection to the recipient
333             // This method waits until the new connection is ready (or not)
```

```
334                 connectToNode(recipients[i]);
335             }
336
337             // Sends the data package to the recipient
338         if (!serviceSearchFailed){
339                     getFrameworkFrontEnd().getGroup().getNode(recipients[i]).getNodeConnection().sendDataPackage(dataPackage);
340         }else{
341             // If the serviceSearch failed, the node must be removed from the group, and groups become synchronized
342             addressesToLostNodes.addElement(recipients[i]);
343         }
344         }
345     }
346         // Removes the nodes that could not be reached to remove these from the group by running a groupsync
347         for(int i=0; i<addressesToLostNodes.size();i++){
348             // Notifies only if the node is not already removed from the local group.
349             // This because a node could have been removed when sending the previous data package and this
350                 // package is sent right after the first one (as in text first and then sync package)
351             if(getFrameworkFrontEnd().getGroup().getNode((String)addressesToLostNodes.elementAt(i))!=null){
352                 getFrameworkFrontEnd().notifyAboutLostNode((String)addressesToLostNodes.elementAt(i));
353             }
354         }
355     }
```

10.3.12 The processSendQueue() method in NodeConnection.java

This method is called by a constantly running separate Thread and sends the first element of the outgoing queue until there are no more data packages to send. The data is sent as bytes over a stream to the recipient.

1. The first step is to transfer an Integer representing the type of the data package so the receiver knows how to treat the incoming data stream.
2. Next an Integer telling the length of the data package's content is sendt. This way the recipient knows how many bytes to read from the incoming data stream.
3. Then the actual content of the data package is transferred byte by byte. When the outgoing queue is empty, a Boolean value (false) is sent to the recipient. In the special case of sending a FilePackage, a FileHandler is used to stream the content of the desired file onto the outgoing stream.

Listing 10.12: The processSendQue() method in NodeConnection.java

```
    /**
     *
     * This method sends datapackages to remote nodes.
     * It processes the que of unsent datapackages.
     * It is called in an infinite loop in the private class OutputThread
     * in this class.
     *
     */
    public synchronized void processSendQueue(){
        if(connection != null){
        if(sendQueue.size() > 0){
          // Retriving the data packages to send from the sendQue
          DataPackage dataPackage = (DataPackage)sendQueue.firstElement();
          sendQueue.removeElement(dataPackage);
          // A byte table holding the data to send
          byte[] data = dataPackage.toSendableFormat();

          try{
            // Opening the output stream if it is not allready open
            if(outputStream == null){
                        outputStream = connection.openDataOutputStream();
                    }

            // Saves a timestamp used to estimate the transfer rate
            long startTime = new Date().getTime();

            // Sending the type of the data package over the steam
                    outputStream.writeInt(dataPackage.getType());

```

```
283                 // Sending the length of the data package over the steam
284             outputStream.writeInt(data.length);
285
286             // Sends the data package in blocks over the stream
287                 // Sending blocks instead of single bytes increases the transfer rate considerably
288             boolean finishedWriting = false;
289             int blockSize = 200;
290             int totalWritten = 0;
291             while(!finishedWriting){
292               // If whats left is less than one blockSize
293               if(data.length - totalWritten < blockSize) blockSize = data.length-totalWritten;
294               byte[] block = new byte[blockSize];
295               // Fills the byte array to be sent
296               for(int i=0; i<blockSize; i++){
297                 block[i] = data[totalWritten];
298                 totalWritten++;
299               }
300               outputStream.write(block);
301
302               if(totalWritten == data.length) finishedWriting = true;
303             }
304
305                 // If the datapackage is a FilePackage we have to send the content
306                 // of the file
307                 long fileSize = 0;
308                 if(dataPackage.getType() == DataPackage.FILE_PACKAGE){
309                   // Opens the file handler
310                   FileHandler fileHandler = new FileHandler(((FilePackage)dataPackage).getFilePath());
311
312                   // Flushes the output stream
313                     outputStream.flush();
314
315                   boolean endOfFile = false;
316                   while(!endOfFile){
317                     try{
318                       byte[] theBytes = fileHandler.readFile();
319                       outputStream.write(theBytes);
320                     }catch(EOFException eofe){
321                       endOfFile = true;
322                       fileHandler.closeFile();
323                     }
324                   }
325                   fileSize = ((FilePackage)dataPackage).getFileSize();
326                 }
327
```

```
328
329                 // Logs a message if the text package was sent successfully
330                 if (dataPackage.getType() == DataPackage.TEXT_PACKAGE ||
331                         dataPackage.getType() == DataPackage.FILE_PACKAGE){
332                     // Estimates the transfer rate of the file
333                     long endTime = new Date().getTime();
334                     long transferTime = (endTime-startTime)/1000;
335                     if (transferTime==0) transferTime = 1;
336                     double kBps = ((double)(data.length+fileSize)/1024)/(double)transferTime;
337
338                     //the code below calculates and rounds off the transfer rate with three decimals
339                     String rate = Double.toString(kBps);
340                     int commaIndex = rate.indexOf(".");
341                     int decimal3 = Integer.parseInt(""+rate.charAt(commaIndex+3)+"");
342                     int decimal4 = Integer.parseInt(""+rate.charAt(commaIndex+4)+"");
343                     rate = rate.substring(0,commaIndex+4);
344                     if (decimal4>=5){
345                         if (decimal3 == 9){
346                             decimal3 = decimal3+1;
347                             rate = rate.substring(0,commaIndex+2);
348                             rate += decimal3;
349                         }
350                         else{
351                             decimal3 = decimal3+1;
352                             rate = rate.substring(0,commaIndex+3);
353                             rate += decimal3;
354                         }
355                     }
356
357                   log.logDataPackage("Finished transfering data to "+
358                             node.getNodeName()+". (Transfer rate was "+rate+"kB/s)");
359                 }
360
361         }catch(IOException ioe){
362             // Because this method is called from within a run() the log has to noitfy the MIDLet of the exception
363             log.logException("NodeConnection.processSendQue()",ioe,true);
364             closeConnection();
365             // Tries to send the datapackage once more
366             currentNetwork.sendDataPackage(dataPackage,dataPackage.getRecipients());
367         }
368         }
369
370
371         // If the queue is not empty, the processing continues
372         try {
```

```
373            Thread.sleep(500);
374        } catch (InterruptedException ie) {
375          // do nothing
376        }
377          try{
378            // Closes the outputstream if the sendQueue is empty
379                  // The connections are re-established when a new datapackage is sent
380                  if (sendQueue.size() == 0){
381                     // Notifies the remote recipient that we are closing the stream
382                     outputStream.writeBoolean(true);
383                     openOutputStream = false;
384                     // Flushes the output stream
385                     outputStream.flush();
386                  }else{
387                      outputStream.writeBoolean(false);
388                      // Flushes the output stream
389                      outputStream.flush();
390                      // Must process the next package
391                    processSendQueue();
392                  }
393        }catch(IOException ioe){
394              // Because this method is called from within a run() the log has to noitfy the MIDLet of the exception
395              log.logException("NodeConnection.processSendQue()2",ioe,true);
396              closeConnection();
397          }
398          }
399      }
```

10.3.13 The processIncomingData() method in NodeConnection.java

This method is the counterpart of the processSendQueue() on the recipient side. It runs in a Thread and continuously checks whether or not a connection to another Node is established. As soon as a connection is present, a stream is opened and it starts listening for incoming data. When some data is received it is decoded into the variables sent from the remote Node. If the received data package is a FilePackage a FileHandler is used to stream the content of the file down to the local file system of the device.

Listing 10.13: The processIncomingData() method in NodeConnection.java

```
    /**
     *
     * This method receives incoming datapackages from remote nodes.
     * It is called in an infinite loop in the private class InputThread
     * in this class.
     *
     */
    public void processIncomingData(){
        if(connection != null){
          boolean connectionFailed = false;
            try{
                if(inputStream == null){
                    inputStream = connection.openDataInputStream();
                }
            }catch(IOException ioe1){
                connectionFailed = true;
                log.logException("NodeConnection.processIncomingData()1",ioe1,true);
                // Opening of streams failed, ergo connection lost
                // Close connection and inform the NodeListener
                try {
                    connection.close();
                    // The connection must be set to null to stop the thread running
                    // this method when the connection has closed
                    connection = null;
                } catch (IOException ioe2){
                    log.logException("NodeConnection.processIncomingData()2",ioe2,true);
                }
            }

            // If an inputstream and an outputstream was successfully opened, a infinite loop starts
            if(!connectionFailed){
                try {
                    while(inputStream != null && connection != null && !connectionFailed){
                        int type = -1;
                        try{
                            // Reads the type of the data package
```

```
132                 type = inputStream.readInt();
133             }catch(IOException ioe){
134                 connectionFailed = true;
135             }
136             // The type of the package determines what should be done with the package
137             switch(type){
138 
139             case(DataPackage.GROUP_SYNC_PACKAGE):
140                 // Reads the length of the incoming package
141                 int byteLength1 = inputStream.readInt();
142                 byte[] bytes1 = new byte[byteLength1];
143                 // Reads the incoming bytes
144                 for(int i=0;i<bytes1.length;i++){
145                   bytes1[i] = inputStream.readByte();
146                 }
147 
148                 // Checks if the sendQue on the sender side is empty
149                 if(inputStream.readBoolean()){
150                   // Closes the connection if the remote node is finished sending all its datapackages
151                   openInputStream = false;
152                 }
153 
154                 GroupSyncPackage groupSyncPackage = new GroupSyncPackage();
155 
156                 // Interprets the content and sets the variables in the groupSyncPackage object
157                 groupSyncPackage.parseBytes(bytes1);
158 
159                 // Notifies the midlet via the frontEnd about the received message.
160                 currentNetwork.getFrameworkFrontEnd().notifyAboutReceivedGroupSyncPackage(groupSyncPackage);
161                 break;
162 
163             case(DataPackage.TEXT_PACKAGE):
164                 // Reads the length of the incoming package
165                 int byteLength2 = inputStream.readInt();
166                 byte[] bytes2 = new byte[byteLength2];
167 
168                 // Reads the incoming bytes in blocks
169                 // Reading blocks increases the transfer rate considerably
170                 boolean finishedReading = false;
171             int blockSize = 200;
172             int totalRead = 0;
173             while(!finishedReading){
174               // If whats left is less than one blockSize
175               if(byteLength2 - totalRead < blockSize) blockSize = byteLength2-totalRead;
176               byte[] block = new byte[blockSize];
```

```
177                 int numberRead = inputStream.read(block,0,blockSize);
178                 // Stores whats read in an array large enough for the whole package
179                 for(int i=0; i<numberRead; i++){
180                   bytes2[totalRead] = block[i];
181                   totalRead++;
182                 }
183                 if(totalRead == byteLength2) finishedReading = true;
184               }
185
186                     // Checks if the sendQue on the sender side is empty
187                     if(inputStream.readBoolean()){
188                       // Closes the connection if the remote node is finished sending all its datapackages
189                       openInputStream = false;
190                     }
191
192                     // Notifies the midlet via the frontEnd about the received message.
193                     TextPackage textPackage = new TextPackage();
194                     textPackage.parseBytes(bytes2);
195
196                     currentNetwork.getFrameworkFrontEnd().notifyAboutReceivedTextPackage(textPackage);
197                     break;
198
199                 case(DataPackage.FILE_PACKAGE):
200                   // Reads the length of the incoming package
201                     int byteLength3 = inputStream.readInt();
202                     byte[] bytes3 = new byte[byteLength3];
203                     // Reads the incoming bytes
204                     for(int i=0;i<bytes3.length;i++){
205                         bytes3[i] = inputStream.readByte();
206                     }
207
208                     // Creates a filePackage based on the received data
209                     FilePackage filePackage = new FilePackage();
210                     filePackage.parseBytes(bytes3);
211                     // Reads the file and writes it to the filesystem
212                     FileHandler fileHandler = new FileHandler(filePackage.getFilePath());
213                     // Fetches the size of the file and sets it in the fileHandler
214                     fileHandler.setFileSize(filePackage.getFileSize());
215
216                     // Checks if the sendQue on the sender side is empty
217                     if(inputStream.readBoolean()){
218                       // Closes the connection if the remote node is finished sending all its datapackages
219                       openInputStream = false;
220                     }
221
```

```
                    boolean endOfFile= false;
                    while(!endOfFile){
                      try{
                        byte[]  theBytes = new byte[fileHandler.getBlockSize()];
                        // Reads data from the inputStream into a byte table
                        int  numberOfBytesRead = inputStream.read(theBytes, 0, fileHandler.getBlockSize());
                        // Writes the bytes to  file
                      fileHandler.writeFile(theBytes, numberOfBytesRead);

                      }catch(EOFException eofe){
                        endOfFile = true;
                        fileHandler.closeFile();
                      }
                    }

                    currentNetwork.getFrameworkFrontEnd().notifyAboutReceivedFilePackage(filePackage);
                    break;

                default:
                    break;
                }
            }

        }catch(IOException ioe) {
            log.logException("NodeConnection.processIncomingData()3", ioe, true);
        }
    }
  }
}
```

10.3.14 The notifyAboutReceivedTextPackage() method in FrameworkFrontEnd.java

When the processIncomingData() method has finished receiving a text package, the framework is notified through this method, which in turn notifies the MIDlet. The MIDlet can now display the text message and the name of the sender.

Listing 10.14: The notifyAboutReceivedTextPackage() method in FrameworkFrontEnd.java

```
/**
 *
 * This method is called from NodeConnection.processIncomingData()
 * whenever a text package is received from a remote node.
 * It processes the package, logs the event, and notifies the midlet.
 *
 * @param textPackage The received text package.
 */
public void notifyAboutReceivedTextPackage(TextPackage textPackage){
    log.logDataPackage("Received text package from "+textPackage.getSender().getNodeName()+".");
    midlet.notifyAboutReceivedTextPackage(textPackage.getSender().getNodeName(), textPackage.getContent());
}
```

Part IV

A Developers Guide to the Peer2Me Framework v2.0

CHAPTER 11

Getting Started with the Peer2Me Framework v2.0

Before providing a user guide to Peer2Me v2.0, we will in this chapter present some central concepts used in this version of the framework. We will also explain how to get started, i.e. which resources that is needed to develop a MIDlet upon Peer2Me v2.0.

11.1 Peer2Me v2.0 Domain Concepts

Knowing the domain concepts is important to be able to understand how Peer2Me v2.0 actually works. Most of the concepts presented here are the same as those presented as central concepts in Peer2Me v1.0 (see Chapter 6.1).

Framework: "A framework is a set of classes that embodies an abstract design for solutions to a family of related problems."

MIDlet: A Java application intended for a CLDC device is called a MIDlet.

Node: A node is a logical representation of a peer.

Group: A group is a collection of nodes running the same MIDlet and communicating using a homogenous network. Every node has a group containing all the nodes it is connected to. Each time a node connects or disconnects, the groups become syncronized on all nodes.

DataPackage: A data package is the entity that can be exchanged between nodes connected in a group. The framework has built in support for three types of data packages; a text package to transfer text, a file package to transfer a file and a group synch package used to synchronize the content of the groups on all nodes.

Figure 11.1 on the next page illustrates a conceptual model of Peer2Me v2.0.

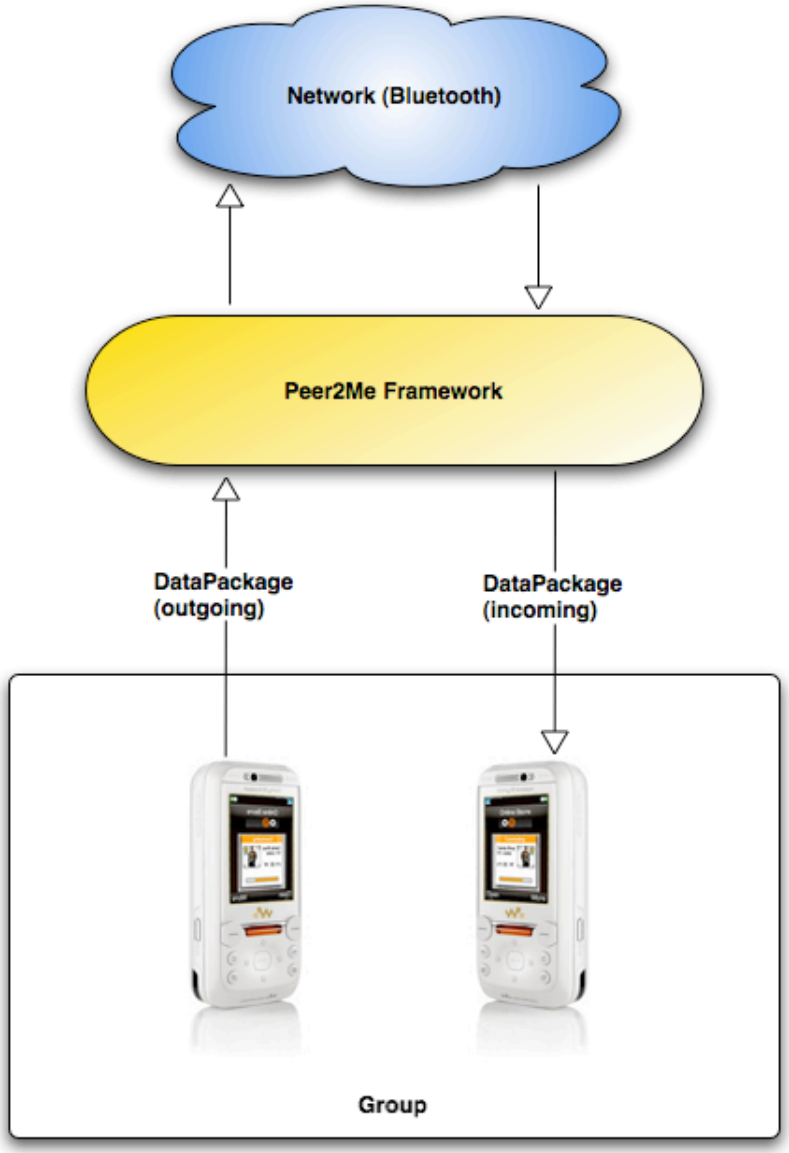

Figure 11.1: Peer2Me conceptual model

11.2 Required Resources

To be able to develop a MIDlet using Peer2Me v2.0, the following resources are needed:

Peer2Me JAR-file: The Peer2Me JAR-file contains the complete framework. Adding this file to the build path in the MIDlet development tool is necessary in order to be able to compile the source code of the MIDlet. In addition, the J2SE Software Development Kit (SDK) must be installed on the computer. This because the *jar.exe* file is needed to read the JAR-file during compilation.

Java2 Standard Edition Software Development Kit (J2SE SDK): The J2SE Software Development Kit (SDK) supports creating J2SE applications. The only reason that this kit is needed, is to get access to the *jar.exe* file in order to read the JAR-file that contains the Peer2Me framework. Alternatively only the *jar.exe* file can be copied and added to the path on the computer.

Java2 Micro edition (J2ME): J2ME is a java edition especially designed for mobile phones, PDAs and other mobile devices. J2ME provides a subset of classes and methods available in the Java2 Standard Edition (J2SE). The Java2 Micro Edition is well described in Chapter 5.1,earlier in this report.

Having the Peer2Me JAR-file, J2SE SDK and J2ME installed on the computer, it should be possible to compile any MIDlet using the Peer2Me framework. The only exception would be if the MIDlet makes use of any special functionality that requires an additional API.

CHAPTER 12

Developing a Peer2Me v2.0 MIDlet

In this chapter we will provide a guide on how to write a MIDlet utilizing the Peer2Me framework. The guide focuses on the Peer2Me specific functions, and leaves out the development of a graphical user interface (GUI). The complete source code of two MIDlets using the Peer2Me framework v2.0 can be found as appendices at the end of this report (Appendix B). While reading this chapter we **strongly** recommend looking up the methods in the Peer2Me v2.0 Javadoc as they get introduced. The Javadoc can be found as Appendix C and contains useful supplementary information about the classes and the methods.

12.1 Initiating the Framework

To get access to the functionality provided by the framework, an instance of the framework must be fetched. To get this instance and to initiate the framework, do the following in the main MIDlet class:

1. Let the main MIDlet class extend class *javax.microedition.midlet.MIDlet*.

2. Import *peer2me.framework.**

3. Let it implement the *peer2me.framework.FrameworkListener* interface. This interface enforces the MIDlet to implement methods used of the framework to send data back to the MIDlet.

4. Run the static method *getInstance* in class *FrameworkFrontEnd* to get a reference of type *Framework*. This can typically be done already in the constructor of the main MIDlet class, and the received reference should be saved globally in the MIDlet as this reference is the one and only reference to the framework from the MIDlet. The input type to *FrameworkFrontEnd.getInstance()* must be a reference to the class implementing the *peer2me.framework.FrameworkListener* interface.

5. When all required inputs are ready, it is time to run the *initFramework* method on the reference fetched in the previous step. This method initiates the framework and causes all internal threads running the different network functionality to start.

12.2 Setting Up a Connection

To search for other devices and to set up a connection, do the following:

1. Run method *startNodeSearch* on the available reference to the framework. This method starts a search for devices running the same MIDlet.

2. If such a device is found, the *notifyAboutFoundNode* method specified by the *FrameworkListener* interface is called to notify the MIDlet. The method *notifyAboutFoundNode* implemented in the MIDlet should display the names of the found nodes in a choice group.

3. After selecting which nodes to connect to from the choice group, the method *connectToNodes* should be called with the addresses of the chosen nodes as input. This method connects the devices in a network.

4. Once a connection is established, the *notifyAboutParticipants* method specified by the *FrameworkListener* interface is called on every connected device to notify the MIDlets about the network participants. The names of the participants should typically become added to a choice group and displayed on each device.

12.3 Sending a Data Package

Sending a data package using Peer2Me v2.0 has become very simple:

1. To send simple text, use the *sendTextPackage* method. This method sends some text over the network. Required input is a list containing the addresses to the recipient nodes, and the text to be sent.

2. When the text package terminates to the recipients, they are alerted by the *notifyAboutReceivedTextPackage* method specified by the *FrameworkListener* interface. The local MIDlet on each recipient should then typically display the received text.

3. To send any kind of file, use the *sendFilePackage* method. This method sends a file over the network. Required input is a list containing the addresses to the recipient nodes, and the path of the file to be sent.

4. When the file package terminates to the recipients, they are alerted by the *notifyAboutReceivedFilePackage* method specified by the *FrameworkListener* interface. The local MIDlet on each recipient should then typically display a text saying that a file has been received.

12.4 Using the Log

The framework has a built-in log that could be very useful during development and testing. The log is used throughout the whole framework, and is filled with different types of status messages informing the user (and/or developer) about what happens during run time. To use the log, do the following:

1. Import *peer2me.util.Log*
2. Fetch the logs by running the static *getLog* method. It is generally very useful to fetch the logs and offer a console containing the content of the logs.
3. New entries can also be added to the log from the MIDlet by using for example the *logDebugInfo* method.

CHAPTER 13

Deploying a Peer2Me MIDlet

This chapter contains a simple guide on how to deploy the MIDlet on a mobile device.

13.1 Creating a MIDlet package

As written in Chapter 5.1.1, a Java application intended for a CLDC device must be formatted into a Java Archive (a JAR-file) to run on the device. JAR files are Java's version of ZIP files, and can in fact be opened with WinZip or WinRar. Usually, a function for creating a JAR-file containing the MIDlet is provided by the development tool, but it can also be created manually using the *jar.exe* file that comes with Java2 Standard Edition Software Development Kit (J2SE SDK). The jar utility program can be run from the command line (DOS prompt or bash for example, depending on your OS). Here is how to create a compressed JAR-file:

jar -cf archiveName.jar file-names-separated-by-space

For additional information on how to use the jar utility program, just type *jar.exe* from the command line, and push the "*enter*" key.

13.2 How to run a MIDlet

To run the MIDlet on a mobile device, the JAR-file containing the compiled MIDlet must be transferred to the device. This can be done using a cable between the computer and the device, or by Bluetooth or IR. Using Bluetooth or IR requires of course that both devices (computer and mobile device) supports the respective wireless network mediums. When the transfer is complete, the JAR-file can be executed and the MIDlet will be installed on the device. The MIDlet is then ready to be run.

Part V

Peer2Me v1.0 vs. Peer2Me v2.0

CHAPTER 14

Comparison of Framework Functionality

In this part we will describe the tests we performed on the Peer2Me v2.0 framework after it was completed. The tests consisted of comparing the redesigned framework with the original one. These tests gave us both quantitative and qualitative data which in turn was used to evaluate the redesign of the Peer2Me framework (see Chapter 18, Evaluating the Redesign) and to answer the Research Questions raised in Chapter 2.1.

The actual testing process was divided into three different parts; a comparison of main functionality (found in the following chapter), a architectural and structural comparison of the original and the redesigned versions of the Peer2Me framework (found in Chapter 15), and comparison of the framework properties (found in Chapter 3).

This chapter contains a presentation of important functionality found in the Peer2Me framework v2.0. If similar functionality also could be found in Peer2Me v1.0 a comparison of the implementation will be made.

14.1 Peer2Me v2.0 Functionality

In the following we will present and describe a selection of functionality found in the Peer2Me framework v2.0;

* Pure peer-to-peer computing, see Chapter 3.3.1.
* Sending text
* Sending files
* Logging
* Detection of lost nodes

* Clean exit

14.1.1 Pure peer-to-peer computing

In Chapter 3.3.1 pure peer-to-peer is defined as:

> " In a pure P2P model it does not exist any central unit (server) responsible for managing or coordinating the services and the resources among the peers in the network."

In Peer2Me v1.0 this is not the case as one of the nodes in the network has to act as a Master handling all the communication between its Slave nodes. This form of peer-to-peer computing is referred to as a hybrid model.

In Peer2Me v2.0 this solution is discarded and replaced with a pure peer-to-peer model. A device running a Peer2Me based MIDlet performs a discovery operation and locates and connects to all or a selection of the discovered devices also running the same MIDlet. Simultaneously all the connected devices are synchronized so they possess knowledge of all the other devices, now acting as nodes in the ad-hoc network. When some data is to be sent, a connection can be established directly between the sender and the recipient(s) rather than through a Master node. This eliminates the Master/Slave concept of Peer2Me v1.0 and removes the potential bottleneck and single point of failure problems of the original framework.

14.1.2 Sending text

Sending of text is the only applicable function of the original Peer2Me framework, but we find the process quite complicated and tedious. In the redesigned version, the text sending is somewhat simplified, but yet very useful. To send a text message the MIDlet needs to call the sendTextMessage() method of the Framework interface. The recipients' network addresses and the actual messages are given as input parameters and the framework handles the rest of the sending process. In Peer2Me v1.0, a message and message parts have to be created, and the entire process is much more time consuming, more complicated and requires several lines of code.

Incoming text is also presented to the MIDlet in a simple way by the Peer2Me v2.0. The notifyAboutReceivedTextPackage() method have to be implemented by all MIDlets due to the implementation of the FrameworkListener interface, and this method is used by the framework to notify the MIDlet about a received text package. The sender's address and the text itself is given as input parameters.

With some additional implementation the text sending can be utilized to serialize and send objects between nodes.

14.1.3 Sending files

This is a completely new feature of the Peer2Me framework v2.0 that allow sending a any kind of files between nodes. The files are fetched from and stored on the local file system of the device running the framework. The files are broken down into blocks of bytes, sent to the receiver(s) and put back together again. The size of the byte blocks are tuned to achieve the highest possible transfer rate.

14.1.4 Logging

The logging functionality is an important feature to allow debugging as there is no console displaying information on this kind of devices. The log also contains a history of all communication and connections established between nodes. The Peer2Me v1.0 has no logging functionality, but features a exception handler and a debug application. There is no consistency in which is used in what situation, whereas Peer2Me v2.0 gathers all information in one location in a readable and descriptive format.

14.1.5 Detection of lost nodes

In a ad-hoc network based on peer-to-peer computing, such as the networks created using the Peer2Me framework, it is crucial to detect cases where one or more nodes are lost. If this is neglected text- and file packages will be lost when one node tries to send to a node that is no longer there. In Peer2Me v2.0 lost nodes are detected by the sender, and if a retry is unsuccessful the lost node is removed from the group. The other nodes of the group is also informed about the event by a synchronization of their groups.

The original Peer2Me framework have no such detection and can only remove a node from a group when it leaves in a controlled way.

14.1.6 Clean exit

When a MIDlet based on the Peer2Me framework shuts down it is important that the framework ”cleans up” as well. All connections and data streams must be closed and removed before the application itself terminates. This way no ”loose threads” are left behind that can compromise the next execution of the same MIDlet. The Peer2Me framework v2.0 has gotten rid of the problems concerning the discovery process experienced in Peer2Me v1.0 when a MIDlet is terminated ungracefully.

CHAPTER 15

Comparison of Code Structure

In this chapter we will describe the improvements we have made to the code structure of the redesigned Peer2Me by comparing it to the structure of similar code found in the original framework. The comparison will consist of code listings of how a particular portion of code is constructed in both the original and the redesigned framework. A description of the listings will point out the improvements and changes we have performed. The last sub chapter will summarize the most important changes and other positive properties of the code.

15.1 Code Samples

The code samples are taken from methods that particularly illustrates how we have redesigned the code to improve the quality and structure. We emphasizes the use of comments, comprehensible variable and method names and generally tidy coding.

15.1.1 Initiation of the framework

These two listings illustrates how the framework is initiated in MIDlets using the original and the redesigned framework.

Listing 15.1: Initiation in Peer2Me v1.0

```
public void init(){
    while(role==null){};

    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("PanIm");

    framework = Framework.getInstance(personalProfile.getNickname(), personalProfile.getFirstname()+" "+
                    personalProfile.getLastname(), " 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);
    showWelcomeMessage();
```

Listing 15.2: Initiation in Peer2Me v2.0

```
// Fetches an instance of the Framework
framework = FrameworkFrontEnd.getInstance(this);
// Initiates the framework
try{
    framework.initFramework(nodeName, midletName, preferredNetwork);
}catch(Exception e){
    append("Error initiating the framework. Please try again.");
}
```

In the MIDlet developed upon the original Peer2Me framework one have to perform a large number of task to initiate the framework, see Listing 15.1. References to a number of classes had to be made, like Framework, Service, Group, and several methods were called.

In the redesigned framework only two lines of code are needed, a instance of the Framework interface is fetched and an initiation is run with user name, MIDlet name and preferred network as input, see Listing 15.2.

15.1.2 Variable Comments

Descriptive and well written comments are important to make the code easy to understand and maintain.

Listing 15.3: Variables in Framework.java (v1.0)

```
  //debug variables
  private boolean debug = false;
  private boolean debug2 = false;

  private Hashtable services = new Hashtable();
  private GroupDiscoveryListener groupDiscoveryListener;
  private Vector messageSubscribers = null;
  private Vector messageQueue;
  private String nodename;
  private String description;
  private Network currentNetwork;
  private Hashtable groups;
  private boolean running;
  private String preferredNetwork;
  private ExceptionHandler exceptionHandler;
```

Listing 15.4: Variables in FrameworkFrontEnd.java (v2.0)

```
    // The instance of the FrameworkFrontEnd returned by the getInstance() method, will be casted to Framework upon return
    private static FrameworkFrontEnd singleton;
    // The midlet that initiated the framework represented by a FrameworkListener instance
    private FrameworkListener midlet;
    // The Network instance of the preferred network
    private Network currentNetwork;
    // The group containing all connected nodes running the same application
    private Group group;
    // The local node
    private Node localNode;
    // A Hastable containing the addresses(key) and names(value) of the nodes found in the discovery process
    private Hashtable foundNodes;

    // A Log instance
    private Log log = Log.getInstance();
```

These code sniplets illustrates how variables are declared and commented in the main framework class of the two versions of the Peer2Me framework. In the redesigned version, every variable has a comment describing the function and use. This is continued throughout all the classes.

15.1.3 Javadoc Comments

To have a complete and good Javadoc of the code is a must for developers using the framework.

Listing 15.5: Javadoc in Framework.java (v1.0)

```
  /**
   * Used for dosconnecting a node.
   *
   * @param node The node to disconnect.
   */
  public void disconnectNode(Node node){
```

Listing 15.6: Javadoc in FrameworkFrontEnd.java (v2.0)

```
132     /**
133      *
134      * This method returns the local representation of the group. It is called from
135      * ConnectionListener.run() or Network.nodeFound() when a remote node is found
136      * and should be added to the group.
137      *
138      * @return The local representation of the group
139      */
140     public Group getGroup(){
```

Every method of the redesigned Peer2Me framework has got a Javadoc comment describing the usage, parameters, exceptions and return values. In some cases the Javadoc also describes where the method is called from. The original framework has some shortcomings of the Javadoc, and quite a few methods lack comments all together.

15.1.4 Method and variable names

Descriptive and well formulated names on methods and variables increases the understanding of their function. This is important for developers creating MIDlets using the framework as well as further development of Peer2Me.

Listing 15.7: allowJoin() method in the GroupMoinitor interface (v1.0)

```
21   /**
22    * Called when a node wishes to join a closed group. Only called if
23    * on the master node.
24    *
25    * @param group The group associated with the join request.
26    * @param node The node sending the request.
27    * @return True if the join is accepted, false otherwise.
28    */
29   public void allowJoin(Group group, Node node);
```

Listing 15.8: sendTextPackage() method in the Framework interface (v2.0)

```
71     /**
72      *
73      * This method sends a text package over the network. When the package
74      * terminates to the recipients , they are alerted by the
75      * notifyAboutReceivedTextPackage() method specified by the
76      * FrameworkListener interface.
77      *
78      * @param recipients A list containing the addresses of the recipient nodes
```

```
     * @param textMessage The text message to be sent
     *
     */
    public void sendTextPackage(String[] recipients, String textMessage);
```

The allowJoin() method is a good example on how the comments and method names of the original framework are somewhat vague, see Listing 15.7.

> * Called when a node wishes to join a closed group. Only called if on the master node.

This description do not explain how the method is supposed work when implemented and where it should be called from. The name of the method is also a bit misleading. Is the method called to allow a remote node to join the group or is it called to attempt to join the group of another node?

In the method sendTextMessage() of the redesigned version of the framework seen in Listing 15.8 the name is descriptive and the input parameters are quite self explaining. The Javadoc comment tells us where the method is called from and what function it serves. The input parameters are also further described in this comment.

15.1.5 Tidy Code

Tidy coding is important to maintain readability and to ensure easy maintenance in the future.

Listing 15.9: The constructor of the RemoteBluetoothNode class (v1.0)

```
public RemoteBluetoothNode(BluetoothNetwork listener, StreamConnection connection) throws BluetoothNodeException{
  this.listener = listener;
  conn = connection;
  try {
    remoteDevice = RemoteDevice.getRemoteDevice(connection);
  } catch (IOException e) {


    throw new BluetoothNodeException("Failed to initialize RemoteBluetoothNode:" + e.toString());


  }
   address = remoteDevice.getBluetoothAddress();



    listener.registerNode(this);

   thread = new Thread(this);
     thread.start();
```

```
21
22        if (debug) Debugger.debug("started thread on this node");
23
24
25  }
```

Listing 15.10: The constructor of the NodeConnection class (v2.0)

```
1  public NodeConnection(StreamConnection connection, Node node){
2
3     // Fetches a instance of the Log
4      log = Log.getInstance();
5      // Sets the connection to connect to and fetches an instance of the currentNetwork
6      this.connection = connection;
7      this.node = node;
8      currentNetwork = Network.getInstance();
9      // Creates the sendQue
10     sendQueue = new Vector();
11
12     // The Input- and OutputStreams shall not do anything before the node is connected
13     // These values are toggled from ConnectionListener.run() and NodeConnection.sendDataPackage()
14     openInputStream = false;
15     openOutputStream = false;
16     // Starts a thread that processes the sendQue
17     outputThread = new OutputThread();
18     outputThread.setPriority(Thread.MAX_PRIORITY);
19     outputThread.start();
20     // Starts a thread constantly listening for incoming datapackages
21     inputThread = new InputThread();
22     inputThread.setPriority(Thread.MAX_PRIORITY);
23     inputThread.start();
24 }
```

The code of the NodeConnection constructor found in the redesigned Peer2Me framework are divided into logical blocks, are compact and has descriptive comments. This eases the work of developers trying to understand the framework with the intention to further improve it.

15.2 Summary

We will now summarize the code structure comparison and point out the improvements we have achieved through the redesign of the Peer2Me framework.

Compact and tidy code The code have been revised to tidy it up and make it more readable. The interface towards the MIDlets is made simpler, located in only one java file and have few methods with descriptive names.

Variable- and Javadoc Comments Comments has been used throughout the entire redesigned framework to describe the usage of variables and methods, and to add Javadoc entries of all methods and classes. The extensive use of comments makes the code easier to comprehend and start using.

Method and Variable Names When using the framework it is crucial that the names of the methods and variables available are self explaining so no confusion about their function occurs. It is time consuming to browse through the code to try and figure out how to use it.

CHAPTER 16

Comparison of Framework Properties

This chapter will aim to describe the comparison of the redesigned as well as the original Peer2Me framework's properties to retrieve the quantitative data needed to evaluate the redesign of the framework (see Chapter 18). The properties we want to compare are; footprint of framework and MIDlets, interface size, complexity, and transfer rate.

Our comparison will be performed with base in the PAN-IM instant messenger MIDlet found in Lund and Norum's master thesis [31] and the Peer2Messenger MIDlet developed by us for the purpose. These two MIDlets will feature the same functionality to ensure a correct comparison. The transfer rate of the original framework was found in the tests performed as a part of our depth study [39].

16.1 The Goal/Question/Metric Approach

These are the goals we stated based on Research Questions 2 and parts of Research Questions 3 in Chapter 2.1 using the the goal definition template found in Chapter 2.2.2. These goals are further refined into questions and metrics used to answer the Research Questions.

Goal 1: Analyze the **the redesigned Peer2Me framework**
for the purpose of **Evaluating the framework and the MIDlets**
with respect to its **footprint and complexity**
from the point of view of the **Developers**
in the context of **Mobile collaborative application development**.

Question 1: Has the redesigned Peer2Me framework got a smaller footprint than the original framework?

Metric: Measure framework footprint.

Question 2: Has a MIDlet built upon the redesign Peer2Me framework got a smaller footprint than a MIDlet built upon the original framework?

Metric: Measure MIDlet footprint.

Question 3: Has the redesigned Peer2Me framework got a smaller interface towards the MIDlets than the original framework?

Metric: Measure the number of relations between a MIDlet and the framework.

Question 4: Is the redesigned Peer2Me framework less complex than the original framework?

Metric: Measure number of packages, number of classes and average class size.

Goal 2: Analyze the **the redesigned Peer2Me framework**
for the purpose of **Evaluating an application built upon the framework**
with respect to its **stability, performance and error rate**
from the point of view of the **Developers**
in the context of **Mobile collaborative application development**.

Question 1: Does the redesigned Peer2Me framework perform better, with respect to transfer rate, than the original framework?

Metric: Measure transfer rate.

16.2 Comparing the Framework Properties

To compare the properties of the redesigned vs. the original framework as well as MIDlets built upon them, we created a Peer2Messenger similar to the PAN-IM presented in the master thesis of Lund and Norum [31]. The two MIDlets had to have the same functionality and GUI to get a correct comparison. To achieve this we had to make some modifications to the PAN-IM MIDlet because it uses persistence to store information about the user. The Modified version of the PAN-IM and the Peer2Messenger have the same functionality and hence they are comparable. The properties we compared in this experiment was:

Framework footprint This property is a factor that can tell us something about the complexity of the frameworks. The idea is that the footprint somewhat reflects the complexity, as a large number of classes will increase the size deployed file containing the framework. A small footprint is also, to some extent, important due to the limited memory available for runtime operations on mobile devices.

MIDlet footprint and interface complexity As in the previous point we presume a connection between the MIDlet size and the complexity. A simplified framework interface will decrease the amount of code needed in the MIDlet to initiate and use the Peer2Me framework. It will also reduce the time a developer needs to start developing working MIDlets.

Framework complexity To make the Peer2Me framework easier to comprehend for developers we have tried to reduce the total number of classes and keep the functionality basic, yet complete. We have also gathered all the methods "visible" to the developers in one interface.

Transfer rate The transfer rate is a crucial property for the framework, as the main focus is communications between two or more portable devices. A limited transfer rate will restrict the possibilities of realtime communication and sending of large files.

Framework Properties			
Properties	Peer2ME v1.0	Peer2Me v2.0	Improvements
Framework Footprint (code lines)	1875	1621	254
Framework Footprint (kB)	47,2kB	37,5kB	9,7kB
Framework Packages	18	6	12
Framework Classes	29	18	11
Framework avg. Class Size (code lines)	65	85	-20
MIDlet Footprint (code lines)	402	321	81
MIDlet Footprint (kB)	9,7kB	9,8kB	-0,1kB
MIDlet Framework Interfaces	4	1	3
MIDlet Framework Interface Methods	8	5	3
MIDlet Framework Class References	6	1	5
Transfer Rate in kB/s	7kB/s	18kB/s	11kB/s

Table 16.1: Framework Properties

We will now review the properties viewed in Table 16.1 to compare the original and the redesigned Peer2Me framework. Each property will be discussed separately and evaluated to determine whether or not the redesign of the framework has improved the specific property.

Framework Footprint The footprint of the framework specific classes has been reduced with about 20.6% (9,7kB), which is significant on small devices like the ones this framework is intended for.

Framework packages The number of packages in the Architecture is reduced to a third of the original. This contributes to reduced complexity and makes the structure easier to comprehend.

Framework classes and average class size Almost 40% reduction in classes shows that redesign has fulfilled the goal of simplifying the framework. The less classes a developer has to relate to and understand the function of, the easier the task of using and further expand the framework will be. The average size of each class has increased with 20 code lines, but this is a natural consequence of the reduction in classes and the classes are still small enough to maintain the developers overview.

MIDlet Footprint The size of the Peer2Messenger MIDlet is not noticeably reduced compared to the PAN-IM MIDlet, although the total size of the deployable JAR file is reduced. This is mainly caused by poor optimization of the code representing the GUI[1] in Peer2Messenger. The actual framework related code of the MIDlet has been minimized.

MIDlet Framework Interfaces and Interface Methods The number of interfaces that has to be implemented by a MIDlet to make use of the framework has been cut from 4 in the original to 1 in the redesigned one. The number of methods that has to be implemented as a result of these interfaces

[1] Graphical User Interface

is reduced from 8 to 5. This means that the job of developing a MIDlet upon the Peer2Me framework has been simplified and easier to comprehend.

MIDlet Framework Class References The number of class references gives an indication of how many classes from the framework a MIDlet has to have a reference to. We have reduced this number from 6 to 1 which reduces the coupling between the Peer2Me framework and the MIDlet. It also eases the task of developing a MIDlet upon the framework because no more than two lines of code is needed to initiate and get a reference to the framework.

Transfer Rate The transfer rate was measured sending a string consisting of the alphabet written out 700 times and we experienced a 157% improvement from the original framework. We also found that the transfer rate increased to more than 30 kB/s when larger amounts of data was transferred (for example a MP3 file of several megagytes). The original framework could not transfer larger data packages, so we can not determine if this increase will occur here as well.

These results and properties will be further discussed in the Evaluation chapter, Chapter 18.

Part VI

Evaluation

CHAPTER 17

Technology Evaluation

In this chapter we will evaluate the technologies utilized in the redesign of the Peer2Me framework and the development tools used in the process. This evaluation will not answer any specific Research Questions, but is provided to give a better overall view of the Peer2Me framework v2.0.

17.1 Mobile Phones and J2ME

The following will evaluate the technology of the mobile phones used in testing during this project, as well as J2ME as a programming platform on such mobile devices.

17.1.1 General Evaluation of Mobile Phones and J2ME

The redesign of the Peer2Me framework is based on SUN Java Wireless Toolkit (WTK) 2.2[1] just as the original is. This version of the toolkit has a wide range of capabilities like support for MIDP 2.0, CLDC 1.1, Bluetooth (JSR-82), FileConnection (JSR-75), Web Services (JSR-172) and 3D graphic (JSR-184). We have kept our focus on MIDP 2.0 and Bluetooth since this are important aspects of the Peer2ME framework. We have also taken advantage of the functionality found in the FileConnection API to enable file sending. The toolkit has performed in a satisfactory way, not causing us any major problems. The mobile phone emulator that comes with the WTK and the emulator found in the Sony Ericsson SDK 2.2.3[2] has also been a helpful tool for testing and debugging. The J2ME implementations of the different mobile phone vendors on the other hand, is still causing a lot of problems. This will be described more thoroughly in the Weaknesses chapter (Chapter 17.1.3).

In this project we have used Sony Ericsson K750 mobile phones as test devices. This phone has proved

[1] http://java.sun.com/products/sjwtoolkit/index.html
[2] developer.sonyericsson.com

itself to be the most satisfactory to develop for and test with, and has only caused minor problems. It is fast, stable and has plenty free memory. The J2ME Bluetooth implementation is also of high quality.

17.1.2 Strengths of Mobile Phones and J2ME

Portability is the main advantage of a combination of technologies like Java in the form of J2ME and portable computing devices, e.g. mobile phones. This portability along with the ability to program fairly powerful applications opens for a wide range of utilizations of this kind of setup. Interaction and collaboration is such a area that is becoming more and more interesting due to the high density of Java enabled mobile devices in affluent countries in the western world.

J2ME is a platform that specially addresses the requirements for program development on small devices like mobile phones and PDAs. This specialization together with the widespread acceptance of J2ME in the consumer electronics has made J2ME one of the leading technologies in this area. This results in a large selection of APIs and available resources that helps developers creating better and more diverse applications. As with all SUN Java platforms J2ME is available for research and development purposes via the Sun Community Source Licensing model which makes it affordable to develop applications on this platform [26].

Bluetooth is becoming more and more widespread as the short range wireless medium of mobile devices. Via the Bluetooth API (JSR-82) the J2ME applications can use the Bluetooth unit of the mobile device to communicate with other devices in the proximity. This opens for the creation of ad hoc networks between devices running the same application, which in turn can be utilized for entertainment, collaboration and awareness purposes. Bluetooth can be used for mobile-to-computer communication as well. J2ME applications can be used to remotely control certain operations on the computer and data can be transferred and synchronized. More about Bluetooth in Chapter 17.2.

With support for file browsing through the FileConnection API (JSR-75) the possibility of sharing files between participants in an ad-hoc network adds a new dimention to the Peer2Me framework v2.0. Accessing and sending files opens for whole new features in the MIDlets developed using the framework.

17.1.3 Weaknesses of Mobile Phones and J2ME

Although the gap between portable and stationary computational devices has shrunk the last few years, there are still several limitations to what can be done on a mobile phone. The relatively low CPU power available restricts the number of calculations per second and hence the possibility to solve complex tasks. The result is that the J2ME applications must be rather lightweight. The limited CPU power is tightly related to the insufficient battery capacity of the devices [11]. This has become one of the biggest challenges for the industry trying to include ever more functionality in smaller and smaller devices.

The means of interaction with the users is still the most restrictive characteristic of portable devices. Both the keyboards and displays are limited in size and degrade the user experience compared to a traditional computer system. Input is often done on numeric keyboards with a multiple key press approach to make writing of text possible. The introduction of a function for predictive text input, like T9 [19], has improved

writing speed for most users, but it is still inferior to a normal QWERTY-keyboard. The display of a typical mobile device[3] features a TFT-screen with 262 144 colors and a 176x220 pixel size. This limits the amount of information that can be presented for the user at one time. Graphics must be kept at a minimum and the focus has to be on the what is important for the usability and usefulness of the application.

J2ME is a stripped down version of J2SE, and because of this it lacks several possibilities that an experienced Java programmer will miss. This makes the transition between the to platforms a bit challenging, but not insuperable. Many of the problems we experienced during development and testing of our Peer2Me application could have been avoided if we could have reviewed some kind of log or console after an error has occurred. This was added as a feature of the Peer2Me framework v2.0 and has proven to be a great help during development and testing.

Another major problem one encounters when developing J2ME applications is that the different mobile device vendors do not implement J2ME uniformly in all devices. The graphical user interface of the different devices is a good example of this with several differences among devices [18]. These differences are mainly connected to grouping and priority of commands, along with layout of forms.

17.2 Bluetooth

We will now evaluate Bluetooth as a wireless medium for communication between two or more mobile devices.

17.2.1 General Evaluation of Bluetooth

The redesigned Peer2Me is intended to support any type of network layer as long as it exists J2ME support and APIs for it. The network layer is kept invisible for the MIDlet running the framework. This way the developer does not have to focus on implementing specific methods to communicate over the network, but rather use available network interfaces. As of today there is only one network layer implemented for the Peer2Me framework, Bluetooth. This is because Bluetooth is the only network technology commonly available in mobile devices at the moment. More details about the Bluetooth technology can be found in Chapter 5.2, Bluetooth.

17.2.2 Strengths of Bluetooth

With exception of WLAN on a few PDA's and Smart Phones, Bluetooth is the only wireless technology commonly available on portable devices like mobile phones today. The use of Bluetooth in wireless hands free units further increase the implementation of the technology into new equipment. The technology is also on its way to become standard on most computers, which opens for wireless communication between mobile and more stationary devices.

To be able to take advantage of Bluetooth's network capabilities in a J2ME application running on a mobile device the JSR-82 Bluetooth API must be present. This API is included on many of the new mobile phones

[3]Sony Ericsson K750

with MIDP 2.0 support. The API enables the J2ME application to communicate with other applications running on other devices via Bluetooth. This gives the possibility to create ad hoc networks, as is the case with the applications based upon the Peer2Me framework.

The experiments we performed concerning range in our depth study [39] proved that the transmitting range of the Bluetooth units in mobile phones is considerably higher than the 10 meters described to be the limitation for Class 2 Bluetooth devices [28]. We were able to sustain a connection between two devices for distances up to about 70 meters both outdoors and indoors and these ranges will make it possible to develop useful applications based upon the Peer2Me framework using Bluetooth as a network layer.

Bluetooth is using a technique called "Frequency Hopping" to increase the quality of the signal. This technique reduces interference and noise, makes the signals harder to intercept and enables higher bandwidth for data transmission. These factors make Bluetooth a relatively stable medium and well suited to be used with the Peer2Me framework v2.0.

17.2.3 Weaknesses of Bluetooth

Through our use and testing of Bluetooth as a wireless network medium we have found the relatively long discovery time to be the main disadvantage. This limits the possibility to create ad hoc networks as the mobile devices have to been in proximity of each other for a long enough period of time before the ad hoc network can be established. The discovery of devices worn by persons passing each other in a corridor is for instance not very likely. A successful creation of an ad hoc network relies on the participants to more or less stationary in range of each other.

The Bluetooth technology has got a theoretical limit of 8 simultaneous connections and this number is quite lower than that on most devices. We have hower overcome this limitation by creating a pure peer-to-peer architecture where the devices are connected only as they send data to each other.

In our depth study [39] we were able to acheive transfer rates of approximately 7kB/s. Although we have increased this to about 20kB/s (see Table 16.1), it is not very high. This restricts the amount of data it is practical to transfer from one device to another and it is mostly suited for transfer of textual messages and small files like images.

As described in Chapter 17.1.3 there are some problems with different mobile device vendors implementing J2ME in a proprietary manner. This affects the Bluetooth API as well; especially in the way the applications are allowed to create a Bluetooth connection. In turn this leads to problems establishing spontaneous as hoc networks as the user has to authenticate any attempt to connect.

17.3 Development Tools

This chapter contains an evaluation of the development tools used in this project.

17.3.1 General Evaluation of the Development Tools

The Eclipse IDE[4] is our preferred environment for writing both code and LaTeX documents as this documentation. To be able to develop J2ME applications we have extended Eclipse with the EclipseME[5] plug-in and SUN's Java Wireless Toolkit[6]. This solutions gives us an easy way of packaging and deploying finished MIDlets as a JAR file ready to execute on J2ME compatible devices. To make this configuration work in Mac OSX we had to do some modifications to the preverifier using elements from a SDK called mpowerplayer[7].

17.3.2 Strengths of the Development Tools

As the Eclipse IDE is Java based and hence is platform independent it is an ideal development environment for this project because of the mix of Windows PCs and Mac OSX macs used. By using this multi platform IDE we eliminated problems that might have occurred if we had used a heterogeneous and OS specific environment.

The EclipseME plug-in gives us several advantages for developing J2ME applications. Combined with the wireless toolkit it has a very useful code assist that makes coding easier and faster. It also features an automatic packaging function that crates a JAR package containing the MIDlet and all the necessary class files. This is far more elegant than the manual approach that includes writing and running an Ant[8] script. The EclipseME have a graphical interface for creating JAD files as well, another helpful element.

Since we are two contributors in this project the built in CVS[9] client found in Eclipse is a very useful tool. I allows to collaborate and work as a team even though we often work from physical different locations. By using CVS to keep track of changes in our code and documentation we can always be sure that we are working with the latest version and we do not have to worry about compromising each other's code. The fact that CVS is using a repository on a remote server also adds an extra level of security against loss of code.

17.3.3 Weaknesses of the Development Tools

The main weakness we have discovered using Eclipse is some stability problems. Eclipse functions can suddenly cease to work and unchanged code that is compiled a minute ago fails to compile. The errors can often be fixed using a combination of restarting, cleaning projects and refreshing form CVS repository.

Another problem we have encountered is incompatibility between the text encoding formats in Mac OSX and Windows (MacRoman and ISO-8859-1). This is resolved by forcing the Mac version of Eclipse to use ISO- 8859-1 for this project.

[4] www.eclipse.org
[5] www.eclipseme.org
[6] http://java.sun.com/products/sjwtoolkit/index.html
[7] mpowerplayer.com
[8] http://ant.apache.org/
[9] http://www.nongnu.org/cvs/

CHAPTER 18

Evaluating the Redesign

In this chapter we will evaluate Peer2Me v2.0 and try to point out strengths and weaknesses of the framework. The evaluation will have a qualitative and subjective character and represents the perceived quality of the redesigned Peer2Me framework. Qualitative data is, as described in Chapter 2.2, expressed in words and gives a richer understanding of the subject in question. This evaluation is used to answer Research Question 1 and the part of Research Question 3 not answered by the results of GQM; Analysis and Interpretation found in Chapter 19. In Research Question 1, we asked:

1. **Does a redesign of the Peer2Me framework improve developers ability to produce J2ME based applications for mobile collaboration?**

18.1 General Evaluation of the Peer2Me Framework v2.0

As the for the original Peer2Me framework is the main purpose for the redesigned version to support development of collaborative mobile applications based on J2ME and any available network protocols. We are still limited to the use of Bluetooth as network medium because this is the only implemented network API for J2ME. Bluetooth is however becoming a widespread technology present on a large range of mobile phones and with recent upgrades it performs quite good as well.

Overall we are satisfied with v2.0 of the framework. We have fulfil out main goals of improving the architecture, simplifying the interface and decreasing the footprint. Our impression is that Peer2Me framework v2.0 is simpler to both learn and use than the original version. Many of the underlying problems and errors are also removed, and the framework appears to be a more reliable and versatile.

18.2 Strengths of the Peer2Me Framework v2.0

The main strength of the redesigned version of the Peer2Me framework is the fact that it now uses a pure peer-to-peer topology in the creation of and communication over ad-hoc networks. This eliminates the bottleneck and single point of failure problem associated with the Master/Slave topology of the Peer2Me v1.0. This solution utilizes the full bandwidth of the network technology and even overcomes the theoretical limitations of simultaneously connected nodes.

As described in Chapter 16, the transfer rate is considerably higher in Peer2Me v2.0 than Peer2Me v1.0. While the perceived transfer rate in v2.0 in a specific case is 18kB/s, it is 7kB/s in v1.0. This is most likely a result of an optimalization of the way bytes are sent and read. In Peer2Me v2.0 bytes a written and read in blocks of a certain size, while in Peer2Me v1.0 bytes are read one by one.

The Peer2Me framework v2.0 encapsulates all network related classes and presents a generic interface towards the MIDlets developed. This approach reduces the time and code required to develop a new MIDlet and does not demand any knowledge of the network technology. New network modules can be added without altering the MIDlet interface at all or having to change the code of existing applications.

A new feature of Peer2Me v2.0 is the inclusion of the FileConnection API (JSR-75). This allow the MIDlets to access the local file system of the device it is executing on. The methods for performing this type of file access are encapsulated by the framework that presents methods for file sending and file list navigation to the MIDlets.

Another new feature of the framework is the detection of lost nodes. If a node in the network tries to send some data to another node that for some reason have disconnected in an irregular manner action is taken. The lost node is removed from the group and all the other participants are synchronized. This way no data is lost or presumed received when this is not the case.

The communication between the Peer2Me framework and the Peer2Me MIDlets is considerably simplified and features two interfaces only, one for MIDlet-to-framework method calls and one for calls in the opposite direction. These interfaces contain only the most basic methods and all non relevant functionality is hidden behind the framework.

The frameworks code are structured and well written with descriptive comments and intuitive method/-variable names. This combined with the developers guide found in Part IV makes the task of creating a new Peer2Me MIDlet manageable to most Java developers.

18.3 Weaknesses of the Peer2Me Framework v2.0

The major weakness of the Peer2Me framework v2.0 is the lack of scatternet support (see Chapter 5.2.6,

Piconets and Scatternets). By adding this kind of network abilities devices that are not in direct network range of each other can communicate via other devices in their individual piconets.

Another drawback is the limited testing we have been able to perform on the finished framework. With more extensive testing we could have eliminated even more potential errors and bugs from the program.

CHAPTER 19

GQM; Analysis and Interpretation

This chapter will be used to answer Research Question 2 and parts of Research Question 3 found in Chapter 2.1:

2. **Does a redesign of the Peer2Me framework reduce the footprint and the complexity of the applications developed as well as the framework itself?**

3. **Will a redesign of the Peer2Me framework increase the performance and decrease the error rate of the applications developed?**

In Chapter 16.1, The Goal/Question/Metric Approach, we have stated two goals that will help us answer question number 2 and some of question 3 found above. We used the Goal Question Metric method to break the goals down into questions and metrics and in this final Analysis and interpretation phase the questions will be answered. Both Goals have entirely quantitative metrics and the data have been gathered through the Comparison of Framework Properties found in Chapter 16.2. When all the answers where found they where used to evaluate if the two main goals have been fulfilled.

19.1 Evaluating Goal 1

This is the first of the goals from the GQM approach found in Chapter 16.1. Created using the GQM template for goals it states:

Goal 1: Analyze the **the redesigned Peer2Me framework**
for the purpose of **Evaluating the framework and the MIDlets**

with respect to its **footprint and complexity**
from the point of view of the **Developers**
in the context of **Mobile collaborative application development**.

We will now answer and evaluate the four questions related to this goal. Based on the answers a conclusion to whether or not the goal is reach will be given.

Question 1: Has the redesigned Peer2Me framework got a smaller footprint than the original framework?

Metric Measure framework footprint.

Peer2Me framework v1.0 footprint 1875 code lines - 47,2kB

Peer2Me framework v2.0 footprint 1531 code lines - 36,1kB

The footprint of the framework specific classes has been reduced with about 16% (11kB). This must be considered a considerably improvement.

Answer The footprint of the framework have been reduced noticeably by the redesign.

Question 2: Has a MIDlet built upon the redesign Peer2Me framework got a smaller footprint than a MIDlet built upon the original framework?

Metric Measure MIDlet footprint.

Peer2Me framework v1.0 MIDlet footprint 402 code lines - 9,7kB

Peer2Me framework v2.0 MIDlet footprint 321 code lines - 9,8kB

The footprint of the MIDlet has been reduced by about 80 code lines, but the size in kB has gotten 0,1kB larger. This could be due to the way the JAR file is packages and the result for a larger MIDlet could have turn out to be different.

Answer The footprint of a MIDlet built upon the Peer2Me framework is slightly reduced.

Question 3: Has the redesigned Peer2Me framework got a smaller interface towards the MIDlets than the original framework?

Metric Measure the number of relations between a MIDlet and the framework.

Peer2Me framework v1.0 relations Interfaces:4 - Class references:6

Peer2Me framework v2.0 relations Interfaces:1 - Class references:1

We have managed to reduce the number of interfaces and class references to one of each. This is the minimal coupling one can achieve and still ensure communication both ways between the framework and the MIDlet.

Answer The coupling between the Peer2Me framework and the MIDlets have decreased considerably from v1.0 to v2.0.

Question 4: Is the redesigned Peer2Me framework less complex than the original framework?

Metric Measure number of packages, number of classes and average class size.

Peer2Me framework v1.0 MIDlet complexity Packages:18 - Classes:29 - Avg. class size: 65 code lines

Peer2Me framework v2.0 MIDlet complexity Packages:6 - Classes:18 - Avg. class size: 85 code lines

Few packages and classes combined with a relatively small average class size indicates that the framework is not very complex.

Answer The numbers tells us that the number of packages is reduced to a third and the number of classes is almost down 40%. The redesigned framework must be considered less complex than the original one.

Conclusion

The answers to Goal1 given above leads us to the conclusion that the goal is reached. The evaluation has shown that the redesign has made the Peer2Me framework both smaller and less complex than it was in it's original form.

19.2 Evaluating Goal 2

This is the evaluation of the second and last goal from the GQM approach found in Chapter 16.1. The goal is:

Goal 2: Analyze the **the redesigned Peer2Me framework**
for the purpose of **Evaluating an application built upon the framework**
with respect to its **stability, performance and error rate**
from the point of view of the **Developers**
in the context of **Mobile collaborative application development**.

The only question of this goal can be read bellow.

Question 1: Does the redesigned Peer2Me framework perform better, with respect to transfer rate, than the original framework?

Metric Measure transfer rate.

Peer2Me framework v1.0 transfer rate 7kB/s

Peer2Me framework v2.0 transfer rate 18kB/s

A high as possible transfer rate is important to ensure practical applications of the framework. High network traffic between multiple devices demands a quite high bandwidth.

Answer The transfer rate of v2.0 of Peer2Me is nearly tripled in comparison with that of the original framework. It is still not a high value, but considering the theoretical limit it is quite satisfactory.

Conclusion

The transfer rate is almost tripled and the goal is absolutely reached.

Part VII

Conclusion

CHAPTER 20

Conclusion

In Chapter 2.1 we stated three research questions. We wanted to find out if a redesign of the Peer2Me framework could improve its usability, performance, modifiability, availability and testability. Throughout this project we have worked according to the methods described in Chapter 2.2, and designed and implemented a new improved version of the Peer2Me framework.

In this chapter we summarize the project results by answering the research questions. The content of Chapter 18 and 19 constitutes a basis for answering these questions.

1. **Does a redesign of the Peer2Me framework improve developers ability to produce J2ME based applications for mobile collaboration?**

   * Yes, our opinion is that Peer2Me v2.0 is considerably easier to make use of than Peer2Me v1.0. This is based on the following improvements in Peer2Me v2.0:
     - Compact and tidy code.
     - Descriptive naming of variables and methods.
     - Well commented code and a comprehensive Javadoc.
     - A simple and intuitive framework interface.

   (a) Is the documentation and the code, with regards to structure and comments, improved sufficiently to decrease the degree of difficulty developing a new application?

   * Yes. The code is made compact and tidy to achieve readability. Comments have been used throughout the entire Peer2Me v2.0 to describe the usage of variables and methods. All methods and classes are well described in a Javadoc to increase developers' ability to quickly comprehend and start using the framework. Methods and variables also have self explaining names, so no confusion about their function occurs.

(b) Does the redesigned architecture increase the developers understanding of the framework's structure, and by this simplify the process of developing a working application?

   * Yes. The new architecture hides the internal structure of the framework for the MIDlet developer. This combined with a simplified framework interface, has made the process of developing a working application considerably easier.

2. **Does a redesign of the Peer2Me framework reduce the footprint and the complexity of the applications developed as well as the framework itself?**

   * Yes. Comparing Peer2Me v1.0 and Peer2Me v2.0 shows a reduction in both footprint and complexity of the developed applications from v1.0 to v2.0.

   (a) Will the redesign of the architecture reduce the footprint of the framework?

      * Yes. The footprint of the framework specific classes has been reduced with about 20%.

   (b) Will improving the interface between the Peer2Me framework and the applications reduce the number of code lines required to develop a working application?

      * Yes. A comparison of two simple instant messenger applications using respectively Peer2Me v1.0 and Peer2Me v2.0, shows a reduction of about 20% in number of code lines.

   (c) Will the the redesign of the architecture reduce the coupling between the Peer2Me framework and the applications?

      * Yes. While MIDlets using Peer2Me v1.0 has to make use four interfaces and six framework classes, MIDlets built upon Peer2Me v2.0 only utilize one interface and one class. This contributes to a reduction of complexity in the MIDlets developed.

3. **Will a redesign of the Peer2Me framework increase the performance and decrease the error rate of the applications developed?**

   * Yes. Experiments have shown that the performance, with regards to transfer rate has improved. The errors discovered in Peer2Me v1.0 are no longer present in Peer2Me v2.0.

   (a) Does the redesigned Peer2Me framework perform better, with respect to transfer rate, than the original framework?

      * Yes. Experiments using a given set of data, shows an improvement of transfer rate by 158% from Peer2Me v1.0 to Peer2Me v2.0.

   (b) Does a revision of the code remove the errors experienced during testing of the original framework?

      * Yes. To our knowledge, all the errors experienced in Peer2Me v1.0 has been eliminated in Peer2Me v2.0.

   (c) Will the introduction of a system for logging the errors as they occur improve the developers ability to correct these errors?

      * Yes, definitively. We experienced that the logging function in Peer2Me v2.0 was very useful during development and testing of MIDlets. We were able to locate and remove errors much faster than we were using Peer2Me v1.0.

CHAPTER 21

Further Work

Although we have announced that Peer2Me v2.0 is robust and simple to use, there are still some work left to do. This chapter contains some short term and long term goals.

21.1 Short Term Goals

To validate the usability of Peer2Me v2.0, it should be tested and evaluated by a group of software developers. This could typically be done by gathering a group of developers to a MIDlet development session. The ideal would be just to give them access to this report and the relevant Javadocs, including the Peer2Me Javadoc, and see if they are able to use the framework as intended. In addition to evaluating the MIDlets developed, it could be useful to make the developers fill out a questionnaire, mapping out things like the level of their skills and their opinions about the Peer2Me framework v2.0.

21.2 Long Term Goals

One long term goal is to add support for other network protocols. Currently, Peer2Me v2.0 only supports Bluetooth as network medium, but support for other network mediums can easily be added as Peer2Me v2.0 already is prepared for this.

Another long term goal is to add support for scatternets. The current mobile phones do not support scatternets at hardware level, which limits the area it is possible to interconnect devices within. Implementing support for scatternets requires a rather complicated dynamic routing algorithm, and is a quite large project in itself.

To evaluate the value of mobile collaborative applications, applications that are stable and easy to use

should be distributed to a large amount of people. By doing this, it would be possible to study how mobile collaborative applications can affect the way people collaborate and communicate.

Part VIII

Appendices

APPENDIX A

Glossary

API An application programming interface is the interface that a computer system or application provides in order to allow requests for service to be made of it by other computer programs, and/or to allow data to be exchanged between them.

CDC The Connected Device Configuration is a framework for J2ME applications targeted at devices with limited resources.

CLDC The Connected Limited Device Configuration is even smaller than the CDC mentioned above and is used for pagers and mobile phones.

CSCW Computer Supported Cooperative Work, a research field that focuses on how computer-based systems can support multiple people working on related tasks.

IEEE The Institute of Electrical and Electronics Engineer is an international non-profit, professional organization for the advancement of technology related to electricity. Consists of 360,000 members in around 175 countries.

GQM Goal Question Metric. The GQM is a method of taking the goals of an empirical study and brake them down into questions and measurement metrics. The GQM method forces scientists to decide upon and define what they actually want to measure before doing it.

J2ME Java 2 Micro Edition, a collection of Java API's targeting smaller consumer electronics like mobile phones, PDA's and so on.

J2SE Java 2 Platform, Standard Edition. J2SE is a complete collection of API's that enables development of Java applications on several platforms of personal computers.

JABWT Java APIs for Bluetooth Wireless Technology. JABWT is a set of standard Java APIs that enable the development of applications in Java conforming to the Bluetooth Specification 1.1.

JSR-82 Java Specification Request 82. A Java API that allows MIDlets to make use of Bluetooth hardware on the device.

MANET Mobile Ad Hoc Networks, a self-configuring network of mobile routers (and associated hosts) connected by wireless links. The routers move at random and organize themselves arbitrarily.

MIDlet A Java program specialized to run on the J2ME virtual machine, often on mobile phones. The main class of the MIDlet has to be a subclass of javax.microedition.midlet.MIDlet and the MIDlet classes have to packaged in a JAR-package. To be runnable the JAR-package has to be preverified by a preverifier.

MIDP Mobile Information Device Profile. The J2ME architecture consists of the Virtual Machine, the CLDC and a so-called profile. MIDP is the only available profile and has reached version 2.0. The profile contains a collection of API's that offers IO-functionality and gaming among other things.

MOWAHS MObile Work Across Heterogeneous Systems. The MOWAHS project is a joint re- search effort by the software engineering and the database technology groups at the Depart- ment of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU).

NTNU The Norwegian University of Science and Technology.

PAN Personal Area Networks are networks connecting users within a personal operating space, typically supporting up to a ten meter range.

P2P The term "peer-to-peer computing" (P2P), refers to the use of computer networks that relies in the computing power and bandwidth of the participants (peers) in the network rather than fixed servers offering resources and services.

Peer2Me Peer2Me is the name of a framework for developing mobile collaborative applications on mobile phones utilizing Personal Area Networks (PANs).

Piconet A piconet is an ad-hoc computer network of devices using Bluetooth technology protocols to allow one master device to interconnect with up to seven active slave devices.

Scatternet When several piconets interconnect a scatternet is created.

WPAN Wireless Personal Area Networks are the wireless equivalent to a PAN.

APPENDIX B

Demo MIDlets

The source code of two MIDlets built upon Peer2Me v2.0.

B.1 Peer2MeDemoMIDlet

```
package midlets;


import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Command;
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.lcdui.ChoiceGroup;
import javax.microedition.lcdui.Choice;

import java.util.Enumeration;
import java.util.Vector;
import java.util.Hashtable;

import peer2me.framework.*;
import peer2me.util.Log;

```

```
23
24 /**
25  *
26  * This class contains a MIDlet using the Peer2Me framework v2.0.
27  * It has a simple GUI and supports sending of both text and files.
28  *
29  * @author Torbjørn Vatn & Steinar A. Hestnes
30  */
31
32 public class Peer2MeDemoMIDlet extends MIDlet implements FrameworkListener{
33
34   // A Log instance
35   Log log = Log.getInstance();
36
37   // The Main GUI of the MIDlet
38   private MainGui mainGui;
39
40   // The Log GUI of the MIDlet
41   private LogGui logGui;
42
43   // The Connect GUI of the MIDlet
44   private ConnectGui connectGui;
45
46   // The Send GUI of the MIDlet
47   private SendGui sendGui;
48
49   // The Send GUI of the MIDlet
50   private FileGui fileGui;
51
52     // A reference to the last displayed gui. Is used to go back.
53     private Displayable lastGui;
54
55   // The Framework instance
56   Framework framework;
57
58   // The variables retrived from the input fields of the GUI
59   private String nodeName;
60
61   // The preferred network of the MIDlet
62   private final String preferredNetwork = "peer2me.network.bluetooth.BluetoothNetwork";
63
64   // The name of the MIDlet
65   private String midletName;
66
67     // A list containing the addresses to the nodes added in the ConnectGui choicegroup
```

```
68      private Hashtable nodeAddressList;
69
70      // A hashtable containing the addresses and names to each connected node.
71      // The names is the keys, and the values is the addresses
72      private Hashtable participatingNodeNames;
73
74
75    /**
76     *
77     * Constructor
78     */
79    public Peer2MeDemoMIDlet(){}
80
81    protected void startApp() throws MIDletStateChangeException{
82      mainGui = new MainGui();
83      logGui = new LogGui();
84      connectGui = new ConnectGui();
85      sendGui = new SendGui();
86      fileGui = new FileGui();
87      // Sets the main gui as the current Displayable
88      showGui(mainGui);
89      // Gets an instance of the Framework
90      framework = FrameworkFrontEnd.getInstance(this);
91      // Setting the name of the MIDlet
92      midletName = "TestMidlet";
93    }
94
95    protected void pauseApp() {}
96
97    /**
98     *
99     * This method is called when the MIDLet is shut down
100    *
101    */
102   protected void destroyApp(boolean arg0) throws MIDletStateChangeException{
103     // Shuts down the framework
104     framework.shutdownFramework();
105     notifyDestroyed();
106   }
107
108   public void notifyAboutException(String location, Exception exception){}
109
110   /**
111    *
112    * This method displays the desired GUI class
```

```
113     *
114     * @param gui
115     */
116    public void showGui(Displayable gui){
117            // Saves a reference to the last displayed gui
118            lastGui = Display.getDisplay(this).getCurrent();
119      // Sets the new gui
120            Display.getDisplay(this).setCurrent(gui);
121      gui.setCommandListener((CommandListener)gui);
122    }
123
124
125      /**
126       *
127       * This method is called by the framework when a node is found. These nodes
128       * are not yet connected in a network. To do this, use the
129       * Framework.connectToNodes() method.
130       *
131       * @param address The network adress of the node
132       * @param remoteNodeName The name of the found remote name
133       */
134      public void notifyAboutFoundNode(String nodeAddress, String remoteNodeName){
135            connectGui.addNode(nodeAddress,remoteNodeName);
136                  showGui(connectGui);
137    }
138
139
140      /**
141       *
142       * This method is called from from the framework to notify the midlet about
143       * the participating devices.
144       *
145       * @param participants A hashtable that contains the names of the participants as unique keys and
146       * the network addresses as values.
147       *
148       */
149      public void notifyAboutParticipants(Hashtable participants){
150            this.participatingNodeNames = participants;
151            sendGui.removeParticipants();
152            sendGui.addParticipants();
153
154            // Shows the gui where we can send datapackages
155            if(participatingNodeNames.size()>0)showGui(sendGui);
156      }
157
```

```
158
159     /**
160      *
161      * This method is called from the framework whenever a text package is
162      * received from a remote node.
163      *
164      * @param senderName The name of the sender
165      * @param textMessage The received text message.
166      */
167     public void notifyAboutReceivedTextPackage(String senderName, String textMessage){
168       String message = senderName+" sent : \n"+textMessage.length()+" characters.\n";
169         sendGui.append(message);
170     }
171
172     /**
173      *
174      * This method is called from the framework whenever a file package is
175      * received from a remote node.
176      *
177      * @param senderName The name of the sender
178      * @param filePath The path to the received file
179      */
180     public void notifyAboutReceivedFilePackage(String senderName, String fileName){
181       String message = senderName+" sent : \n"+fileName+"\n";
182         sendGui.append(message);
183     }
184
185   /**
186    *
187    * This class is a private GUI class used to display GUI elements for this
188    * MIDlet
189    *
190    * @author Torbjørn Vatn & Steinar A. Hestnes
191    */
192   private class MainGui extends Form implements CommandListener{
193
194
195     // The OK Command
196     private Command ok;
197     private Command displayLog;
198     private Command search;
199     private Command exit;
200
201
202     // The text filed use to input a name
```

```
203     TextField text;
204
205     /**
206      *
207      * Constructor Extends Form to be able to display different GUI elements
208      *
209      */
210     public MainGui(){
211       super(midletName);
212       ok = new Command("Ok", Command.OK, 0);
213         displayLog = new Command("View Log", Command.OK, 4);
214         search = new Command("Search", Command.OK, 1);
215       exit = new Command("Exit", Command.EXIT,0);
216       text = new TextField("Name", "Per Tome", 30, TextField.ANY);
217
218       append(text);
219
220       addCommand(ok);
221             addCommand(displayLog);
222             addCommand(exit);
223     }
224
225
226     /**
227      * This method is called when the CommandListener detects the use of a Command
228      *
229      * @param command The command used
230      * @param disp The displayable
231      *
232      */
233     public void commandAction(Command command, Displayable disp){
234
235       if (command == ok){
236         // Fetches the input from the gui
237         nodeName = text.getString();
238         deleteAll();
239                 removeCommand(ok);
240                 append("Your name is "+nodeName+". \n\nPress Search if you want to discover other devices.\n\n");
241         addCommand(search);
242
243                 try{
244                     framework.initFramework(nodeName, midletName, preferredNetwork);
245                 }catch(Exception e){
246                     append("Error initiating the framework. Please try again." +e);
247                 }
```

```
248
249
250                 }else  if (command == search){
251             try{
252                     framework.startNodeSearch();
253                     append("Searching for devices running Peer2MeDemoMIDlet\n");
254                 }catch(Exception e){
255                     append("Could not start a search for other devices. Please try again."+e);
256                 }
257
258         }else  if (command == displayLog){
259                 logGui.append("Exception log:\n");
260                 logGui.append(log.getLog(Log.EXCEPTION_LOG));
261                 logGui.append("\n***************\n");
262
263                 logGui.append("Data package log:\n");
264                 logGui.append(log.getLog(Log.DATA_PACKAGE_LOG));
265                 logGui.append("\n***************\n");
266
267                 logGui.append("Connection log:\n");
268                 logGui.append(log.getLog(Log.CONNECTION_LOG));
269                 logGui.append("\n***************\n");
270
271                 logGui.append("Debug log:\n");
272                 logGui.append(log.getLog(Log.DEBUG_LOG));
273                 logGui.append("\n***************\n");
274                 showGui(logGui);
275             }
276
277         else  if (command == exit){
278           try{
279             destroyApp(true);
280           }catch (MIDletStateChangeException msce) {
281             // This exception is ignored because the unconditional attribute of the
282             // destroyApp() method is true.
283           }
284         }
285
286       }
287
288     }
289
290     /**
291      *
292      * This class is a private GUI class to display Log elements for this MIDlet
```

```
293     *
294     * @author Torbjørn Vatn & Steinar A. Hestnes
295     */
296    private class LogGui extends Form implements CommandListener{
297
298      // The log menu Commands
299      private Command showExceptionLog;
300      private Command showConnectionLog;
301      private Command showDataPackageLog;
302      private Command showDebugLog;
303      private Command displayLog;
304      private Command hide;
305      private Command exit;
306
307
308      public LogGui(){
309        super("Log");
310        // The log menu Commands
311        showExceptionLog = new Command("Exception log", Command.ITEM, 4);
312        showConnectionLog = new Command("Connection log", Command.ITEM, 4);
313        showDataPackageLog = new Command("Data package log", Command.ITEM, 4);
314        showDebugLog = new Command("Debug log", Command.ITEM, 4);
315        displayLog = new Command("Full log", Command.ITEM, 1);
316        hide = new Command("Hide log", Command.OK, 0);
317        exit = new Command("Exit", Command.EXIT,0);
318
319        addCommand(showExceptionLog);
320        addCommand(showDataPackageLog);
321        addCommand(showConnectionLog);
322        addCommand(showDebugLog);
323            addCommand(displayLog);
324        addCommand(hide);
325        addCommand(exit);
326      }
327
328      /**
329       * This method is called when the CommandListener detects the use of a Command
330       *
331       * @param command The command used
332       * @param disp The displayable
333       *
334       */
335      public void commandAction(Command command, Displayable disp){
336        if (command == showExceptionLog){
337          this.deleteAll();
```

```
338             append(log.getLog(Log.EXCEPTION_LOG));
339         }
340
341         else if (command == showConnectionLog){
342                 deleteAll ();
343             append(log.getLog(Log.CONNECTION_LOG));
344         }
345
346         else if (command == showDataPackageLog){
347                 deleteAll ();
348                 append(log.getLog(Log.DATA_PACKAGE_LOG));
349         }
350
351         else if (command == showDebugLog){
352                 deleteAll ();
353                 append(log.getLog(Log.DEBUG_LOG));
354         }
355
356         else if (command == displayLog){
357                 deleteAll ();
358                 logGui.append("Exception log:\n");
359                 logGui.append(log.getLog(Log.EXCEPTION_LOG));
360                 logGui.append("\n***************\n");
361
362                 logGui.append("Data package log:\n");
363                 logGui.append(log.getLog(Log.DATA_PACKAGE_LOG));
364                 logGui.append("\n***************\n");
365
366                 logGui.append("Connection log:\n");
367                 logGui.append(log.getLog(Log.CONNECTION_LOG));
368                 logGui.append("\n***************\n");
369
370                 logGui.append("Debug log:\n");
371                 logGui.append(log.getLog(Log.DEBUG_LOG));
372                 logGui.append("\n***************\n");
373         }
374
375         else if (command == hide){
376             deleteAll ();
377             showGui(lastGui);
378         }
379
380         else if (command == exit){
381             try{
382                 destroyApp(true);
```

```
383         }catch (MIDletStateChangeException msce) {
384           // This exception is ignored because the unconditional attribute of the
385           // destroyApp() method is true.
386         }
387       }
388     }
389   }
390   /**
391    *
392    * This class is a private GUI class to display Connect elements for this MIDlet
393    *
394    * @author Torbjørn Vatn & Steinar A. Hestnes
395    */
396   private class ConnectGui extends Form implements CommandListener{
397
398     // The commands
399     private Command back;
400     private Command displayLog;
401     private Command connect;
402     private Command exit;
403
404     //The ChoiceGroup of
405     private ChoiceGroup nodes;
406
407
408         /**
409          *
410          * Constructor
411          */
412     public ConnectGui(){
413       super("Nodes");
414
415       // The ChoiceGroup
416       nodes = new ChoiceGroup("Choose the node(s) to connect to",Choice.MULTIPLE);
417
418               nodeAddressList = new Hashtable();
419
420       // The Commands
421       back = new Command("Back",Command.BACK,1);
422       displayLog = new Command("Display log",Command.ITEM,4);
423       connect = new Command("Connect",Command.ITEM,0);
424       exit  = new Command("Exit", Command.EXIT,0);
425
426       addCommand(back);
427             addCommand(connect);
```

```
428         addCommand(displayLog);
429         addCommand(exit);
430         // Adds the ChoiceGroup
431         append(nodes);
432 
433     }
434 
435     /**
436      *
437      * This method adds a node name to the ChoiceGroup
438      *
439      * @param address The address of the remote node
440          * @param remoteNodeName The name of the remote node
441      */
442     public void addNode(String nodeAddress, String remoteNodeName){
443             nodeAddressList.put(remoteNodeName,nodeAddress);
444             nodes.append(remoteNodeName,null);
445     }
446 
447     /**
448      * This method is called when the CommandListener detects the use of a Command
449      *
450      * @param command The command used
451      * @param disp The displayable
452      *
453      */
454     public void commandAction(Command command, Displayable disp) {
455 
456         if (command == back){
457             showGui(lastGui);
458         }
459 
460             else if (command == displayLog){
461                 logGui.append("Exception log:\n");
462                 logGui.append(log.getLog(Log.EXCEPTION_LOG));
463                 logGui.append("\n***************\n");
464 
465                 logGui.append("Data package log:\n");
466                 logGui.append(log.getLog(Log.DATA_PACKAGE_LOG));
467                 logGui.append("\n***************\n");
468 
469                 logGui.append("Connection log:\n");
470                 logGui.append(log.getLog(Log.CONNECTION_LOG));
471                 logGui.append("\n***************\n");
472 
```

```
473                 logGui.append("Debug log:\n");
474                 logGui.append(log.getLog(Log.DEBUG_LOG));
475                 logGui.append("\n***************\n");
476                 showGui(logGui);
477         }
478
479                 else if (command == connect){
480         // Fetches all the selected elements in the ChoiceGroup
481                 Vector addressVector = new Vector();
482         for (int i=0; i<nodes.size(); i++){
483                         if (nodes.isSelected(i)){
484                           addressVector.addElement((String)nodeAddressList.get(nodes.getString(i)));
485                         }
486         }
487         String[] addresses = new String[addressVector.size()];
488         addressVector.copyInto(addresses);
489         if (addresses.length == 0) append("Please chose a recipient!");
490         else framework.connectToNodes(addresses);
491         }
492
493                 else if (command == exit){
494               try{
495                 destroyApp(true);
496               }catch (MIDletStateChangeException msce) {
497                 // This exception is ignored because the unconditional attribute of the
498                 // destroyApp() method is true.
499               }
500             }
501
502     }
503
504 }
505 /**
506  *
507  * This class is a private GUI class to display Send elements for this MIDlet
508  *
509  * @author Torbjørn Vatn & Steinar A. Hestnes
510  */
511 private class SendGui extends Form implements CommandListener{
512
513     // The commands
514     private Command displayLog;
515     private Command sendTextPackage;
516     private Command sendFilePackage;
517     private Command clearScreen;
```

```
518     private Command exit;
519
520     // The ChoiceGroup
521     private ChoiceGroup connectedNodes;
522     private TextField factor;
523
524     /**
525      *
526      * Constructor
527      */
528     public SendGui(){
529       super("Send datapackage");
530
531       // Creating  the Commands
532       sendTextPackage = new Command("SendTextPackage",Command.ITEM,1);
533       sendFilePackage = new Command("SendFilePackage",Command.ITEM,2);
534       displayLog = new Command("View log",Command.ITEM,3);
535       clearScreen  = new Command("Clear screen",Command.ITEM,4);
536       exit  = new Command("Exit", Command.EXIT,0);
537
538       // The ChoiceGroup containing the connected Nodes
539       connectedNodes = new ChoiceGroup("Choose recipients", Choice.MULTIPLE);
540       factor  = new TextField("Alphabet factor","700",5,TextField.ANY);
541
542       // Adding the elements
543             addCommand(sendTextPackage);
544             addCommand(sendFilePackage);
545       addCommand(displayLog);
546       addCommand(clearScreen);
547       addCommand(exit);
548       append(connectedNodes);
549       append(factor);
550     }
551
552         /**
553          *
554          * This method add the connected participants to the connected ChoiceGroup
555          *
556          */
557         public void addParticipants(){
558             Enumeration names = participatingNodeNames.keys();
559             while(names.hasMoreElements()){
560                 connectedNodes.append((String)names.nextElement(),null);
561             }
562         }
```

```
563
564         /**
565          *
566          * This method removes all connected participants to the connected ChoiceGroup
567          *
568          */
569         public void removeParticipants(){
570             connectedNodes.deleteAll();
571         }
572
573     /**
574      * This method is called when the CommandListener detects the use of a Command
575      *
576      * @param command The command used
577      * @param disp The displayable
578      *
579      */
580     public void commandAction(Command command, Displayable disp){
581
582         if (command == sendTextPackage){
583                 Vector recipientNodes = new Vector();
584                 for (int i=0; i<connectedNodes.size(); i++){
585                     if (connectedNodes.isSelected(i)){
586                         // Gets the address to the recipient node
587                         recipientNodes.addElement((String)participatingNodeNames.get(connectedNodes.getString(i)));
588                     }
589                 }
590                 String [] recipientAddresses = new String[recipientNodes.size ()];
591                 recipientNodes.copyInto(recipientAddresses);
592
593                 String alfa = "abcdefghijklmnopqrstuvwxyz";
594                 String message = "";
595                 for (int i=0;i<Integer.parseInt(factor.getString ()); i++){
596                     message += alfa;
597                 }
598
599                 if (recipientAddresses.length!=0)framework.sendTextPackage(recipientAddresses,message);
600                 else append("Please choose a recipient!\n");
601
602         }else if (command == sendFilePackage){
603
604             Vector recipientNodes = new Vector();
605                 for (int i=0; i<connectedNodes.size(); i++){
606                     if (connectedNodes.isSelected(i)){
607                         // Gets the address to the recipient node
```

```
608                     recipientNodes.addElement((String)participatingNodeNames.get(connectedNodes.getString(i)));
609                 }
610             }
611             String[] recipientAddresses = new String[recipientNodes.size()];
612             recipientNodes.copyInto(recipientAddresses);
613
614             if(recipientAddresses.length!=0){
615               // Sets the recipents and switches GUI
616               fileGui.setRecipients(recipientAddresses);
617               fileGui.fillList();
618         showGui(fileGui);
619             }else append("Please choose a recipient!\n");
620
621           }else if(command == displayLog){
622             logGui.append("Exception log:\n");
623             logGui.append(log.getLog(Log.EXCEPTION_LOG));
624             logGui.append("\n***************\n");
625
626             logGui.append("Data package log:\n");
627             logGui.append(log.getLog(Log.DATA_PACKAGE_LOG));
628             logGui.append("\n***************\n");
629
630             logGui.append("Connection log:\n");
631             logGui.append(log.getLog(Log.CONNECTION_LOG));
632             logGui.append("\n***************\n");
633
634             logGui.append("Debug log:\n");
635             logGui.append(log.getLog(Log.DEBUG_LOG));
636             logGui.append("\n***************\n");
637             showGui(logGui);
638
639           }else if(command == clearScreen){
640             sendGui.deleteAll();
641             // Creates a clean SendGui
642             sendGui = new SendGui();
643             sendGui.addParticipants();
644             // Displays it
645             showGui(sendGui);
646
647         }else if(command == exit){
648           try{
649             destroyApp(true);
650           }catch (MIDletStateChangeException msce) {
651             // This exception is ignored because the unconditional attribute of the
652             // destroyApp() method is true.
```

```
653         }
654       }
655     }
656   }
657
658   /**
659    *
660    * This class is a private GUI class to display FileSend elements for this MIDlet
661    *
662    * @author Torbjørn Vatn & Steinar A. Hestnes
663    */
664   private class FileGui extends List implements CommandListener{
665
666     // The commands
667     private Command displayLog;
668     private Command sendFilePackage;
669     private Command updateList;
670     private Command exit;
671
672     // The root of the file system on the SE K750/W800
673     private String root = "c:/other/Peer2Me/";
674
675     // The root of the file system on the SE Emulator
676     //private String root = "peer2me/";
677
678     // A Enumeration containing the files in the root directory
679     Enumeration files;
680
681     // The addresses of the recipients of the chosen file
682     String[] recipientAddresses;
683
684     /**
685      *
686      * Constructor
687      *
688      */
689     public FileGui(){
690       super("Send filePackage", List.EXCLUSIVE);
691
692       // Creating the elements
693       displayLog = new Command("View log",Command.ITEM,4);
694       sendFilePackage = new Command("Send File",Command.ITEM,2);
695       updateList = new Command("Update List",Command.ITEM,2);
696       exit = new Command("Exit", Command.EXIT,0);
697
```

```
698         // Adds the commands
699         addCommand(displayLog);
700         addCommand(sendFilePackage);
701         addCommand(updateList);
702         addCommand(exit);
703     }
704 
705     /**
706      *
707      * This method fills the fileList List with the names of the files
708      * located in root.
709      *
710      */
711     public void fillList (){
712         // Fills the files Enumeration
713         files = framework.getFileList(root);
714 
715         if( files != null){
716             // Fills the filesList List
717             while ( files .hasMoreElements()) {
718                 append((String) files .nextElement(), null );
719             }
720         }
721     }
722 
723     /**
724      *
725      * This method sets the recipientAddresses
726      *
727      * @param recipientAddresses The recipientAddresses to set
728      */
729     public void setRecipients(String[] recipientAddresses){
730         this.recipientAddresses = recipientAddresses;
731     }
732 
733     /**
734      * This method is called when the CommandListener detects the use of a Command
735      *
736      * @param command The command used
737      * @param disp The displayable
738      *
739      */
740     public void commandAction(Command command, Displayable disp){
741 
742         if(command == sendFilePackage){
```

```
743
744         framework.sendFilePackage(recipientAddresses,root+getString(getSelectedIndex()));
745         fileGui . deleteAll ();
746         showGui(sendGui);
747
748       }
749
750       else  if (command == updateList){
751          fillList  ();
752       }
753
754
755       else  if (command == displayLog){
756                 logGui.append("Exception log:\n");
757                 logGui.append(log.getLog(Log.EXCEPTION_LOG));
758                 logGui.append("\n***************\n");
759
760                 logGui.append("Data package log:\n");
761                 logGui.append(log.getLog(Log.DATA_PACKAGE_LOG));
762                 logGui.append("\n***************\n");
763
764                 logGui.append("Connection log:\n");
765                 logGui.append(log.getLog(Log.CONNECTION_LOG));
766                 logGui.append("\n***************\n");
767
768                 logGui.append("Debug log:\n");
769                 logGui.append(log.getLog(Log.DEBUG_LOG));
770                 logGui.append("\n***************\n");
771                 showGui(logGui);
772             }
773
774       else  if (command == exit){
775         try{
776           destroyApp(true);
777         }catch (MIDletStateChangeException msce) {
778           // This exception is ignored because the unconditional attribute of the
779           // destroyApp() method is true.
780         }
781       }
782
783     }
784   }
785 }
```

B.2 Peer2Messenger

```java
package peer2Messenger;


import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.TextField;
import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.Choice;

import java.util.Enumeration;
import java.util.Vector;
import java.util.Hashtable;

import peer2me.framework.*;
import peer2me.util.Log;



/**
 *
 * This class contains a MIDlet built upon Peer2Me v2.0.
 * The MIDlet is a basic chat application.
 *
 * @author Torbjørn Vatn & Steinar A. Hestnes
 */

public class Peer2Messenger extends MIDlet implements FrameworkListener{

  // A Log instance
  Log log = Log.getInstance();

  // The Main GUI of the MIDlet
  private MainGui mainGui;

  // The Log GUI of the MIDlet
  private LogGui logGui;

  // The Connect GUI of the MIDlet
```

```
44    private ConnectGui connectGui;
45
46    // The Send GUI of the MIDlet
47    private SendGui sendGui;
48
49      // A reference to the last displayed gui. Is used to go back.
50      private Displayable lastGui;
51
52    // The Framework instance
53    Framework framework;
54
55    // The variables retrived from the input fields of the GUI
56    private String nodeName;
57
58    // The preferred network of the MIDlet
59    private final String preferredNetwork = "peer2me.network.bluetooth.BluetoothNetwork";
60
61    // The name of the MIDlet
62    private String midletName;
63
64      // A list containing the addresses to the nodes added in the ConnectGui choicegroup
65      private Hashtable nodeAddressList;
66
67      // A hashtable containing the addresses and names to each connected node.
68      // The names is the keys, and the values is the addresses
69      private Hashtable participatingNodeNames;
70
71
72    /**
73     *
74     * Constructor
75     */
76    public Peer2Messenger(){}
77
78    protected void startApp() throws MIDletStateChangeException{
79      mainGui = new MainGui();
80      logGui = new LogGui();
81      connectGui = new ConnectGui();
82      sendGui = new SendGui();
83      // Sets the main gui as the current Displayable
84      showGui(mainGui);
85      // Gets an instance of the Framework
86      framework = FrameworkFrontEnd.getInstance(this);
87      // Setting the name of the MIDlet
88      midletName = "TestMidlet";
```

```
89
90      }
91
92      protected void pauseApp() {}
93
94      /**
95       *
96       * This method is called when the MIDLet is shut down
97       *
98       */
99      protected void destroyApp(boolean arg0) throws MIDletStateChangeException{
100
101       // Shuts down the framework
102       framework.shutdownFramework();
103
104       notifyDestroyed();
105     }
106
107     public void notifyAboutException(String location, Exception exception){}
108
109     /**
110      *
111      * This method displays the desired GUI class
112      *
113      * @param gui
114      */
115     public void showGui(Displayable gui){
116           // Saves a reference to the last displayed gui
117           lastGui = Display.getDisplay(this).getCurrent();
118       // Sets the new gui
119           Display.getDisplay(this).setCurrent(gui);
120       gui.setCommandListener((CommandListener)gui);
121     }
122
123
124       /**
125        *
126        * This method is called by the framework when a node is found. These nodes
127        * are not yet connected in a network. To do this, use the
128        * Framework.connectToNodes() method.
129        *
130        * @param address The network adress of the node
131        * @param remoteNodeName The name of the found remote name
132        */
133       public void notifyAboutFoundNode(String nodeAddress, String remoteNodeName){
```

```
134         connectGui.addNode(nodeAddress,remoteNodeName);
135             showGui(connectGui);
136 }
137
138
139
140     /**
141      *
142      * This method is called from from the framework to notify the midlet about
143      * the participating devices.
144      *
145      * @param participants A hashtable that contains the names of the participants as unique keys and
146      * the network addresses as values.
147      *
148      */
149     public void notifyAboutParticipants(Hashtable participants){
150         this.participatingNodeNames = participants;
151         sendGui.removeParticipants();
152         sendGui.addParticipants();
153
154         // Shows the gui where we can send datapackages
155         if(participatingNodeNames.size()>0)showGui(sendGui);
156     }
157
158
159     /**
160      *
161      * This method is called from the framework whenever a text package is
162      * received from a remote node.
163      *
164      * @param senderName The name of the sender
165      * @param textMessage The received text message.
166      */
167     public void notifyAboutReceivedTextPackage(String senderName, String textMessage){
168       String message = "\n"+senderName+" says: \n"+textMessage;
169         sendGui.append(message);
170     }
171
172     /**
173      *
174      * This method is called from the framework whenever a file package is
175      * received from a remote node.
176      *
177      * @param senderName The name of the sender
178      * @param filePath The path to the received file
```

```
179      */
180     public void notifyAboutReceivedFilePackage(String senderName, String fileName){}
181
182   /**
183    *
184    * This class is a private GUI class used to display GUI elements for this
185    * MIDlet
186    *
187    * @author Torbjørn Vatn & Steinar A. Hestnes
188    */
189   private class MainGui extends Form implements CommandListener{
190
191
192     // The OK Command
193     private Command ok;
194     private Command displayLog;
195     private Command search;
196     private Command exit;
197
198
199     // The text filed use to input a name
200     TextField text;
201
202     /**
203      *
204      * Constructor Extends Form to be able to display different GUI elements
205      *
206      */
207     public MainGui(){
208       super(midletName);
209       ok = new Command("Ok", Command.OK, 0);
210         displayLog = new Command("View Log", Command.OK, 4);
211         search = new Command("Search", Command.OK, 1);
212       exit = new Command("Exit", Command.EXIT,0);
213       text = new TextField("Name", "Per Tome", 30, TextField.ANY);
214
215       append(text);
216
217       addCommand(ok);
218           addCommand(displayLog);
219           addCommand(exit);
220     }
221
222
223     /**
```

```
224     * This method is called when the CommandListener detects the use of a Command
225     *
226     * @param command The command used
227     * @param disp The displayable
228     *
229     */
230    public void commandAction(Command command, Displayable disp){
231
232      if (command == ok){
233        // Fetches the input from the gui
234        nodeName = text.getString();
235        deleteAll();
236                  removeCommand(ok);
237                  append("Your name is "+nodeName+". \n\nPress Search if you want to discover other devices.\n\n");
238        addCommand(search);
239
240                 try{
241                     framework.initFramework(nodeName, midletName, preferredNetwork);
242                 }catch(Exception e){
243                     append("Error initiating the framework. Please try again.");
244                 }
245
246
247               }else if (command == search){
248          try{
249                     framework.startNodeSearch();
250                     append("Searching for devices running Peer2Messenger\n");
251                 }catch(Exception e){
252                     append("Could not start a search for other devices. Please try again."+e);
253                 }
254
255      }else if (command == displayLog){
256                 logGui.append("Exception log:\n");
257                 logGui.append(log.getLog(Log.EXCEPTION_LOG));
258                 logGui.append("\n***************\n");
259
260                 logGui.append("Data package log:\n");
261                 logGui.append(log.getLog(Log.DATA_PACKAGE_LOG));
262                 logGui.append("\n***************\n");
263
264                 logGui.append("Connection log:\n");
265                 logGui.append(log.getLog(Log.CONNECTION_LOG));
266                 logGui.append("\n***************\n");
267
268                 logGui.append("Debug log:\n");
```

```
269                 logGui.append(log.getLog(Log.DEBUG_LOG));
270                 logGui.append("\n***************\n");
271                 showGui(logGui);
272             }
273
274         else if (command == exit){
275           try{
276             destroyApp(true);
277           }catch (MIDletStateChangeException msce) {
278             // This exception is ignored because the unconditional attribute of the
279             // destroyApp() method is true.
280           }
281         }
282
283       }
284
285     }
286
287     /**
288      *
289      * This class is a private GUI class to display Log elements for this MIDlet
290      *
291      * @author Torbjørn Vatn & Steinar A. Hestnes
292      */
293     private class LogGui extends Form implements CommandListener{
294
295       // The log menu Commands
296       private Command showExceptionLog;
297       private Command showConnectionLog;
298       private Command showDataPackageLog;
299       private Command showDebugLog;
300       private Command displayLog;
301       private Command hide;
302       private Command exit;
303
304
305       public LogGui(){
306         super("Log");
307         // The log menu Commands
308         showExceptionLog = new Command("Exception log", Command.ITEM, 4);
309         showConnectionLog = new Command("Connection log", Command.ITEM, 4);
310         showDataPackageLog = new Command("Data package log", Command.ITEM, 4);
311         showDebugLog = new Command("Debug log", Command.ITEM, 4);
312         displayLog = new Command("Full log", Command.ITEM, 1);
313         hide = new Command("Hide log", Command.OK, 0);
```

```
314         exit = new Command("Exit", Command.EXIT,0);
315
316         addCommand(showExceptionLog);
317         addCommand(showDataPackageLog);
318         addCommand(showConnectionLog);
319         addCommand(showDebugLog);
320                 addCommand(displayLog);
321                 addCommand(hide);
322         addCommand(exit);
323     }
324
325     /**
326      * This method is called when the CommandListener detects the use of a Command
327      *
328      * @param command The command used
329      * @param disp The displayable
330      *
331      */
332     public void commandAction(Command command, Displayable disp){
333         if (command == showExceptionLog){
334             this.deleteAll();
335             append(log.getLog(Log.EXCEPTION_LOG));
336         }
337
338         else if (command == showConnectionLog){
339                     deleteAll();
340             append(log.getLog(Log.CONNECTION_LOG));
341         }
342
343         else if (command == showDataPackageLog){
344                     deleteAll();
345                     append(log.getLog(Log.DATA_PACKAGE_LOG));
346         }
347
348         else if (command == showDebugLog){
349                     deleteAll();
350                     append(log.getLog(Log.DEBUG_LOG));
351         }
352
353         else if (command == displayLog){
354                     deleteAll();
355                     logGui.append("Exception log:\n");
356                     logGui.append(log.getLog(Log.EXCEPTION_LOG));
357                     logGui.append("\n***************\n");
358
```

```
359                 logGui.append("Data package log:\n");
360                 logGui.append(log.getLog(Log.DATA_PACKAGE_LOG));
361                 logGui.append("\n***************\n");
362
363                 logGui.append("Connection log:\n");
364                 logGui.append(log.getLog(Log.CONNECTION_LOG));
365                 logGui.append("\n***************\n");
366
367                 logGui.append("Debug log:\n");
368                 logGui.append(log.getLog(Log.DEBUG_LOG));
369                 logGui.append("\n***************\n");
370         }
371
372         else if (command == hide){
373           deleteAll ();
374           showGui(lastGui);
375         }
376
377         else if (command == exit){
378           try{
379             destroyApp(true);
380           }catch (MIDletStateChangeException msce) {
381             // This exception is ignored because the unconditional attribute of the
382             // destroyApp() method is true.
383           }
384         }
385       }
386     }
387     /**
388      *
389      * This class is a private GUI class to display Connect elements for this MIDlet
390      *
391      * @author Torbjørn Vatn & Steinar A. Hestnes
392      */
393     private class ConnectGui extends Form implements CommandListener{
394
395       // The commands
396       private Command back;
397       private Command displayLog;
398       private Command connect;
399       private Command exit;
400
401       //The ChoiceGroup of
402       private ChoiceGroup nodes;
403
```

```
404
405         /**
406          *
407          * Constructor
408          */
409     public ConnectGui(){
410       super("Nodes");
411
412       // The ChoiceGroup
413       nodes = new ChoiceGroup("Choose the node(s) to connect to",Choice.MULTIPLE);
414
415               nodeAddressList = new Hashtable();
416
417       // The Commands
418       back = new Command("Back",Command.BACK,1);
419       displayLog = new Command("Display log",Command.ITEM,4);
420       connect = new Command("Connect",Command.ITEM,0);
421       exit  = new Command("Exit", Command.EXIT,0);
422
423       addCommand(back);
424             addCommand(connect);
425       addCommand(displayLog);
426       addCommand(exit);
427       // Adds the ChoiceGroup
428       append(nodes);
429
430     }
431
432     /**
433      *
434      * This method adds a node name to the ChoiceGroup
435      *
436      * @param address The address of the remote node
437          * @param remoteNodeName The name of the remote node
438      */
439     public void addNode(String nodeAddress, String remoteNodeName){
440             nodeAddressList.put(remoteNodeName,nodeAddress);
441             nodes.append(remoteNodeName,null);
442     }
443
444     /**
445      * This method is called when the CommandListener detects the use of a Command
446      *
447      * @param command The command used
448      * @param disp The displayable
```

```
449     *
450     */
451    public void commandAction(Command command, Displayable disp) {
452
453      if (command == back){
454        showGui(lastGui);
455      }
456
457            else if (command == displayLog){
458                logGui.append("Exception log:\n");
459                logGui.append(log.getLog(Log.EXCEPTION_LOG));
460                logGui.append("\n***************\n");
461
462                logGui.append("Data package log:\n");
463                logGui.append(log.getLog(Log.DATA_PACKAGE_LOG));
464                logGui.append("\n***************\n");
465
466                logGui.append("Connection log:\n");
467                logGui.append(log.getLog(Log.CONNECTION_LOG));
468                logGui.append("\n***************\n");
469
470                logGui.append("Debug log:\n");
471                logGui.append(log.getLog(Log.DEBUG_LOG));
472                logGui.append("\n***************\n");
473                showGui(logGui);
474      }
475
476              else if (command == connect){
477        // Fetches all the selected elements in the ChoiceGroup
478              Vector addressVector = new Vector();
479        for (int i=0; i<nodes.size(); i++){
480                    if (nodes.isSelected(i)){
481                      addressVector.addElement((String)nodeAddressList.get(nodes.getString(i)));
482                    }
483        }
484        String[] addresses = new String[addressVector.size()];
485        addressVector.copyInto(addresses);
486        if (addresses.length == 0) append("Please chose a recipient!");
487        else framework.connectToNodes(addresses);
488      }
489
490              else if (command == exit){
491            try{
492              destroyApp(true);
493            }catch (MIDletStateChangeException msce) {
```

```
494                 // This exception is ignored because the unconditional attribute of the
495                 // destroyApp() method is true.
496               }
497             }
498
499     }
500
501   }
502   /**
503    *
504    * This class is a private GUI class to display Send elements for this MIDlet
505    *
506    * @author Torbjørn Vatn & Steinar A. Hestnes
507    */
508   private class SendGui extends Form implements CommandListener{
509
510     // The commands
511     private Command displayLog;
512     private Command sendTextPackage;
513     private Command exit;
514
515     // The ChoiceGroup
516     private ChoiceGroup connectedNodes;
517
518     private TextField input;
519
520     /**
521      *
522      * Constructor
523      */
524     public SendGui(){
525       super("Write a message");
526
527       // Creating the Commands
528       sendTextPackage = new Command("Send",Command.ITEM,1);
529       displayLog = new Command("View log",Command.ITEM,4);
530       exit = new Command("Exit", Command.EXIT,0);
531       input = new TextField("Message","",200,TextField.ANY);
532
533       // The ChoiceGroup containing the connected Nodes
534       connectedNodes = new ChoiceGroup("Choose recipients", Choice.MULTIPLE);
535
536       // Adding the elements
537             addCommand(sendTextPackage);
538       addCommand(displayLog);
```

```
539         addCommand(exit);
540         append(connectedNodes);
541         append(input);
542     }
543 
544         /**
545          *
546          * This method add the connected participants to the connected ChoiceGroup
547          *
548          */
549         public void addParticipants(){
550             Enumeration names = participatingNodeNames.keys();
551             while(names.hasMoreElements()){
552                 connectedNodes.append((String)names.nextElement(),null);
553             }
554         }
555 
556         /**
557          *
558          * This method removes all connected participants to the connected ChoiceGroup
559          *
560          */
561         public void removeParticipants(){
562             connectedNodes.deleteAll();
563         }
564 
565     /**
566      * This method is called when the CommandListener detects the use of a Command
567      *
568      * @param command The command used
569      * @param disp The displayable
570      *
571      */
572     public void commandAction(Command command, Displayable disp){
573 
574         if (command == sendTextPackage){
575                 Vector recipientNodes = new Vector();
576                 for (int i=0; i<connectedNodes.size(); i++){
577                     if (connectedNodes.isSelected(i)){
578                         // Gets the address to the recipient node
579                         recipientNodes.addElement((String)participatingNodeNames.get(connectedNodes.getString(i)));
580                     }
581                 }
582                 String [] recipientAddresses = new String[recipientNodes.size ()];
583                 recipientNodes.copyInto(recipientAddresses);
```

```
584
585                 if (recipientAddresses.length!=0)framework.sendTextPackage(recipientAddresses,input.getString());
586                 else  append("Please choose a recipient!\n");
587
588
589             }else  if (command == displayLog){
590                 logGui.append("Exception log:\n");
591                 logGui.append(log.getLog(Log.EXCEPTION_LOG));
592                 logGui.append("\n***************\n");
593
594                 logGui.append("Data package log:\n");
595                 logGui.append(log.getLog(Log.DATA_PACKAGE_LOG));
596                 logGui.append("\n***************\n");
597
598                 logGui.append("Connection log:\n");
599                 logGui.append(log.getLog(Log.CONNECTION_LOG));
600                 logGui.append("\n***************\n");
601
602                 logGui.append("Debug log:\n");
603                 logGui.append(log.getLog(Log.DEBUG_LOG));
604                 logGui.append("\n***************\n");
605                 showGui(logGui);
606             }
607
608              else  if (command == exit){
609         try{
610           destroyApp(true);
611         }catch (MIDletStateChangeException msce) {
612           // This exception is  ignored because the unconditional attribute  of the
613           // destroyApp() method is true.
614         }
615       }
616     }
617   }
618 }
```

APPENDIX C

Peer2Me v2.0 Javadoc

The Javadoc on Peer2Me v2.0. Due to limited support for generating Javadoc in LaTeX format, the readability is not optimal in this appendix. For maximum readability, we recommend the digital version of the Peer2Me v2.0 Javadoc which is bundled with the Peer2Me v2.0 JAR file.

C.1 Package peer2me.domain

Package Contents

Classes

DataPackage 198
This class is the super class of the different type of packages that can be sent between nodes in the network.

FilePackage 200
This class represents a data package containing metadata about a file of some sort that should be sent over the network.

Group 202
This class represents a group of nodes running the same service (MIDlet).

GroupSyncPackage 205
A GroupSyncPackage is a package used internally in the framework to synchronize the groups containing the participants.

Node 206
This class represents a node in the network.

TextPackage 209
This class represents a data package containing text that should be sent over the network.

C.1.1 Class DataPackage

This class is the super class of the different type of packages that can be sent between nodes in the network. It contains the attributes that are common for all types of data packages. These are the address of the sender and the address(es) to the recipiant(s) of the DataPackage. Currently, there exists three types of data packages.

Declaration

public abstract class DataPackage
extends java.lang.Object java.lang.Object

All known subclasses

TextPackagepeer2me.domain.TextPackage, GroupSyncPackagepeer2me.domain.GroupSyncPackage, FilePackagepeer2me.domain.FilePackage

Field summary

FILE_PACKAGE
GROUP_SYNC_PACKAGE
log
TEXT_PACKAGE

Constructor summary

DataPackage(int) Constructor used to create an empty DataPackage object to fill with the parseBytes() method
DataPackage(int, Node, String[]) Constructor

Method summary

getRecipients() This method returns all the recipients of this data package
getSender() This method returns the sender of this data package
getType() This method returns an int indicating the type of data package
parseBytes(byte[]) This method parses the content of the byte array (byte[]) back into a DataPackage object
setRecipients(String[]) This method sets the nodes to receive this package
setSender(Node) This method sets the sender of this data package
toSendableFormat() This method transforms this data package into a byte array (byte[]) that is possible to send over a network stream

Fields

* public peer2me.util.Log **log**

* public static final int **GROUP_SYNC_PACKAGE**
* public static final int **TEXT_PACKAGE**
* public static final int **FILE_PACKAGE**

Constructors

* **DataPackage**
  `public` **DataPackage**`( int` **type** `)`
  - **Description**
    Constructor used to create an empty DataPackage object to fill with the parseBytes() method
  - **Parameters**
    * `type` – The type of the DataPackage
* **DataPackage**
  `public` **DataPackage**`( int` **type**`, Node` **sender**`, java.lang.String[]` **recipients** `)`
  - **Description**
    Constructor
  - **Parameters**
    * `type` – The type specifying the type of data package
    * `sender` – A node object representing the sender node
    * `recipients` – The addresses to the recipients of the data package

Methods

* **getRecipients**
  `public java.lang.String[]` **getRecipients**`( )`
  - **Description**
    This method returns all the recipients of this data package
  - **Returns** – recipients The addresses to the recipients of this package
* **getSender**
  `public Node` **getSender**`( )`
  - **Description**
    This method returns the sender of this data package
  - **Returns** – sender The node that sends this package
* **getType**
  `public int` **getType**`( )`

- **Description**

  This method returns an int indicating the type of data package

- **Returns** – type An int indicating the type of data package

* **parseBytes**
  `public abstract void` **parseBytes**`( byte[] data )`

  - **Description**

    This method parses the content of the byte array (byte[]) back into a DataPackage object

  - **Parameters**

    * `data` – The byte[] containing the data representing the DataPackage object

* **setRecipients**
  `public void` **setRecipients**`( java.lang.String[] recipients )`

  - **Description**

    This method sets the nodes to receive this package

  - **Parameters**

    * `recipients` – The addresses to the nodes that shall receive this package

* **setSender**
  `public void` **setSender**`( Node sender )`

  - **Description**

    This method sets the sender of this data package

  - **Parameters**

    * `sender` – The node that sends this package

* **toSendableFormat**
  `public abstract byte[]` **toSendableFormat**`( )`

  - **Description**

    This method transforms this data package into a byte array (byte[]) that is possible to send over a network stream

  - **Returns** – The byte[] representation of the data package

C.1.2 Class FilePackage

This class represents a data package containing metadata about a file of some sort that should be sent over the network. The package contains the file path and length of the file to transfer, so that the receiver can handle the incoming stream of data and transform it back into a copy of the file.

Declaration

public class FilePackage
extends peer2me.domain.DataPackage peer2me.domain.DataPackage

Constructor summary

FilePackage() Constructor used to create an empty FilePackage object to fill with the parseBytes() method
FilePackage(Node, String[], String) Constructor

Method summary

getFilePath() This method returns the file path of this FilePackage
getFileSize() This method returns the file size of this FilePackage
parseBytes(byte[]) This method parses the content of the byte array (byte[]) back into a FilePackage object
toSendableFormat() This method transforms this file package into a byte array (byte[]) that is possible to send over a network stream

Constructors

* **FilePackage**
  `public` **FilePackage**`( )`
  - **Description**
    Constructor used to create an empty FilePackage object to fill with the parseBytes() method
* **FilePackage**
  `public` **FilePackage**`( Node` **sender**`, java.lang.String[]` **recipients**`, java.lang.String` **filePath** `)`
  - **Description**
    Constructor
  - **Parameters**
    * `sender` – A node object representing the sender node
    * `recipients` – The addresses to the recipients of the file package
    * `filePath` – The path of the file to be sent

Methods

* **getFilePath**
  `public java.lang.String` **getFilePath**`( )`
  - **Description**
    This method returns the file path of this FilePackage

- **Returns** – The file path

* **getFileSize**
  `public long` **getFileSize**`( )`

  - **Description**

    This method returns the file size of this FilePackage

  - **Returns** – The file size

* **parseBytes**
  `public void` **parseBytes**`( byte[] data )`

  - **Description**

    This method parses the content of the byte array (byte[]) back into a FilePackage object

  - **Parameters**

    * `data` – The byte[] containing the data representing the FilePackage object

* **toSendableFormat**
  `public byte[]` **toSendableFormat**`( )`

  - **Description**

    This method transforms this file package into a byte array (byte[]) that is possible to send over a network stream

  - **Returns** – The byte[] representation of the file package

Members inherited from class `peer2me.domain.DataPackage` peer2me.domain.DataPackage

* `public static final` **FILE_PACKAGE**
* `public String` **getRecipients**`( )`
* `public Node` **getSender**`( )`
* `public int` **getType**`( )`
* `public static final` **GROUP_SYNC_PACKAGE**
* `public` **log**
* `public abstract void` **parseBytes**`( byte[] data )`
* `public void` **setRecipients**`( java.lang.String[] recipients )`
* `public void` **setSender**`( Node sender )`
* `public static final` **TEXT_PACKAGE**
* `public abstract byte` **toSendableFormat**`( )`

C.1.3 Class Group

This class represents a group of nodes running the same service (MIDlet). All connected nodes in the ad hoc network are participants in the group. Participants can be added and removed, and a list of all the participants can be retreived.

Declaration

public class Group
extends java.lang.Object java.lang.Object

Constructor summary

Group() Constructor.

Method summary

addParticipant(Node) This method adds a node to the group as a participant.
getNode(String) This method returns a node with the address specified as input
getParticipatingNodeNames(FrameworkFrontEnd) This method returns a list containg the names (as keys) of the nodes participating in this group.
getParticipatingNodes() This method returns a list containing the nodes participating in this group.
removeAllParticipants() This method removes all participating nodes.
removeParticipant(String) This method removes a participating node.
shutdownGroup() This method closes the NodeConnection of all the participating nodes, and removes all nodes from the group.

Constructors

* **Group**
  `public` **Group**`( )`
  - **Description**
    Constructor. Creates a new Group. A group is created in FrameworkFrontEnd.initFramework().

Methods

* **addParticipant**
  `public void` **addParticipant**`( Node` **node** `)`
  - **Description**
    This method adds a node to the group as a participant.
  - **Parameters**
    * `node` – The node to add as a participant.
* **getNode**
  `public Node` **getNode**`( java.lang.String` **address** `)`
  - **Description**
    This method returns a node with the address specified as input
  - **Parameters**

    * `address` – The address of the node to get
  - **Returns** – A node with the address specified as input

* **getParticipatingNodeNames**
  `public java.util.Hashtable` **getParticipatingNodeNames**`( peer2me.framework.FrameworkFrontEnd` **frameworkFrontEnd** `)`
  - **Description**
    This method returns a list containg the names (as keys) of the nodes participating in this group. The addresses are stored as values. It is called from FrameworkFrontEnd.notifyAboutParticipants().
  - **Returns** – A list containing the nodes participating in this group. The node name is the key and the address is the value

* **getParticipatingNodes**
  `public java.util.Hashtable` **getParticipatingNodes**`( )`
  - **Description**
    This method returns a list containing the nodes participating in this group. The address is the key to find the Node.
  - **Returns** – A list containing the nodes participating in this group. The address is the key and the node name is the value

* **removeAllParticipants**
  `public void` **removeAllParticipants**`( )`
  - **Description**
    This method removes all participating nodes. It is used to clear the group before it is updated by a groupSyncPackage received from a remote node.

* **removeParticipant**
  `public void` **removeParticipant**`( java.lang.String` **address** `)`
  - **Description**
    This method removes a participating node.
  - **Parameters**
    * `address` – The address of the node to remove from this group

* **shutdownGroup**
  `public void` **shutdownGroup**`( )`
  - **Description**
    This method closes the NodeConnection of all the participating nodes, and removes all nodes from the group. It is called from the MIDlet via FrameworkFrontEnd.shutdownFramework() when all network connections should be closed.

C.1.4 Class GroupSyncPackage

A GroupSyncPackage is a package used internally in the framework to synchronize the groups containing the participants. The participant performing the groupsync uses its own group as content of the package. All the receivers synchronizes their groups based on the information found in the GroupSyncPackage.

Declaration

public class GroupSyncPackage
extends peer2me.domain.DataPackage peer2me.domain.DataPackage

Constructor summary

GroupSyncPackage() Constructor used to create an empty GroupSyncPackage object to fill with the parseBytes() method

GroupSyncPackage(Node, String[], Node[]) Constructor

Method summary

getParticipants() This method returns a list of the nodes that are participating in the network (group)

parseBytes(byte[]) This method parses the content of the byte array (byte[]) back into a GroupSyncPackage object

toSendableFormat() This method transforms this groupsync package into a byte array (byte[]) that is possible to send over a network stream

Constructors

* **GroupSyncPackage**
  `public` **GroupSyncPackage**`( )`
  - **Description**
    Constructor used to create an empty GroupSyncPackage object to fill with the parseBytes() method
* **GroupSyncPackage**
  `public` **GroupSyncPackage**`( Node` **sender**`, java.lang.String[]` **recipients**`, Node[]` **participatingNodes** `)`
  - **Description**
    Constructor
  - **Parameters**
    * `sender` – A node object representing the sender node
    * `recipients` – The addresses to the recipients of the groupsync package
    * `participatingNodes` – A hashtable with node addresses as keys and names as values

Methods

* **getParticipants**
  `public Node[] getParticipants( )`
  - **Description**
    This method returns a list of the nodes that are participating in the network (group)
  - **Returns** – A list of participating nodes
* **parseBytes**
  `public void parseBytes( byte[] data )`
  - **Description**
    This method parses the content of the byte array (byte[]) back into a GroupSyncPackage object
  - **Parameters**
    * `data` – The byte[] containing the data representing the GroupSyncPackage object
* **toSendableFormat**
  `public byte[] toSendableFormat( )`
  - **Description**
    This method transforms this groupsync package into a byte array (byte[]) that is possible to send over a network stream
  - **Returns** – The byte[] representation of the groupsync package

Members inherited from class `peer2me.domain.DataPackage` peer2me.domain.DataPackage

* `public static final` **FILE_PACKAGE**
* `public String` **getRecipients**`( )`
* `public Node` **getSender**`( )`
* `public int` **getType**`( )`
* `public static final` **GROUP_SYNC_PACKAGE**
* `public` **log**
* `public abstract void` **parseBytes**`( byte[] data )`
* `public void` **setRecipients**`( java.lang.String[] recipients )`
* `public void` **setSender**`( Node sender )`
* `public static final` **TEXT_PACKAGE**
* `public abstract byte` **toSendableFormat**`( )`

C.1.5 Class Node

This class represents a node in the network. It contains information like the name of the node and its network address. A node also owns a nodeConnection object listening for- and processing incoming and outgoing data packages.

Declaration

public class Node
extends java.lang.Object java.lang.Object

Constructor summary

Node(String, StreamConnection) Constructor.
Node(String, String) Constructor.
Node(String, String, StreamConnection) Constructor.

Method summary

getAddress() This method returns the node address
getNodeConnection() This method returns the NodeConnection owned by this node
getNodeName() This method returns the name of the node
restoreNode(String) This method restores a node with the properties specified in the given input string.
setNodeConnection(StreamConnection) This method sets the connection to this remote node.
setNodeName(String) This method sets the name of the node
startNodeConnection() This method creates a nodeConnection running two threads.

Constructors

* **Node**
  `public` **Node**`( java.lang.String` **address**`, javax.microedition.io.StreamConnection` **connection** `)`
  - **Description**
    Constructor. Creates a new Node. This constructor is used when a node is created to represent a remote device on the node which was DISCOVERED during a search. In this case, only the address is known. In addition, a StreamConnection object containing a connection to this remote device exists. The constructor is called from the run() method in ConnectionListener.
  - **Parameters**
    * `address` – The node network address
    * `connection` – The connection to this remote node

* **Node**
  `public` **Node**`( java.lang.String` **nodeName**`, java.lang.String` **address** `)`
  - **Description**
    Constructor. Creates a new Node. This constructor is used when a node is created to represent the LOCAL device. In this case, nodeName and address are known. The constructor is called from FrameworkFrontEnd.initFramework().

- **Parameters**
  - `nodeName` – The name of the node
  - `address` – The node network address

* **Node**
  `public` **Node**`( java.lang.String` **nodeName**`, java.lang.String` **address**`, javax.microedition.io.StreamConnection` **connection** `)`
  - **Description**
    Constructor. Creates a new Node. This constructor is used when a node is created to represent a remote device on the node which INITIATED the search. In this case, name and address is known. In addition, a StreamConnection object containing a connection to this remote device exists. The constructor is called from the nodeFound() method in the Network subclass.
  - **Parameters**
    - `nodeName` – The name of the node
    - `address` – The node network address
    - `connection` – The connection to this remote node

Methods

* **getAddress**
  `public java.lang.String` **getAddress**`( )`
  - **Description**
    This method returns the node address
  - **Returns** – The node network address

* **getNodeConnection**
  `public peer2me.network.NodeConnection` **getNodeConnection**`( )`
  - **Description**
    This method returns the NodeConnection owned by this node
  - **Returns** – nodeConnection This nodes NodeConnection

* **getNodeName**
  `public java.lang.String` **getNodeName**`( )`
  - **Description**
    This method returns the name of the node
  - **Returns** – The nodeName

* **restoreNode**
  `public static Node` **restoreNode**`( java.lang.String` **nodeString** `)`

- **Description**

  This method restores a node with the properties specified in the given input string.

- **Parameters**

  - `nodeString` – A string containing node properties (name:address)

* **setNodeConnection**
  `public void` **setNodeConnection**`( javax.microedition.io.StreamConnection` **connection** `)`

  - **Description**

    This method sets the connection to this remote node. It is called from Network.nodeFound().

  - **Parameters**

    - `connection` – The connection to this remote node

* **setNodeName**
  `public void` **setNodeName**`( java.lang.String` **nodeName** `)`

  - **Description**

    This method sets the name of the node

  - **Parameters**

    - `nodeName` – The name of the node

* **startNodeConnection**
  `public void` **startNodeConnection**`( )`

  - **Description**

    This method creates a nodeConnection running two threads. One of the threads listens for incoming data packages, and the other processes outgoing data packages. It is only used when this node object represents a remote node.

C.1.6 Class TextPackage

This class represents a data package containing text that should be sent over the network.

Declaration

public class TextPackage
extends peer2me.domain.DataPackage peer2me.domain.DataPackage

Constructor summary

TextPackage() Constructor used to create an empty TextPackage object to fill with the parseBytes() method

TextPackage(Node, String[], String) Constructor

Method summary

getContent() This method returns the text content of this TextPackage

parseBytes(byte[]) This method parses the content of the byte array (byte[]) back into a TextPackage object

toSendableFormat() This method transforms this text package into a byte array (byte[]) that is possible to send over a network stream

Constructors

* **TextPackage**
  `public` **TextPackage**`( )`
  - **Description**
    Constructor used to create an empty TextPackage object to fill with the parseBytes() method
* **TextPackage**
  `public` **TextPackage**`( Node` **sender**`, java.lang.String[]` **recipients**`, java.lang.String` **content** `)`
  - **Description**
    Constructor
  - **Parameters**
    * `sender` – A node object representing the sender node
    * `recipients` – The addresses to the recipients of the text package
    * `content` – The String to be sent

Methods

* **getContent**
  `public java.lang.String` **getContent**`( )`
  - **Description**
    This method returns the text content of this TextPackage
  - **Returns** – The content
* **parseBytes**
  `public void` **parseBytes**`( byte[]` **data** `)`
  - **Description**
    This method parses the content of the byte array (byte[]) back into a TextPackage object
  - **Parameters**
    * `data` – The byte[] containing the data representing the TextPackage object

* **toSendableFormat**
  `public byte[] toSendableFormat( )`

  - **Description**

    This method transforms this text package into a byte array (byte[]) that is possible to send over a network stream

  - **Returns** – The byte[] representation of the text package

Members inherited from class `peer2me.domain.DataPackage` peer2me.domain.DataPackage

* `public static final` **FILE_PACKAGE**
* `public String` **getRecipients**`( )`
* `public Node` **getSender**`( )`
* `public int` **getType**`( )`
* `public static final` **GROUP_SYNC_PACKAGE**
* `public` **log**
* `public abstract void` **parseBytes**`( byte[]` **data** `)`
* `public void` **setRecipients**`( java.lang.String[]` **recipients** `)`
* `public void` **setSender**`( Node` **sender** `)`
* `public static final` **TEXT_PACKAGE**
* `public abstract byte` **toSendableFormat**`( )`

C.2 Package peer2me.framework

Package Contents

Interfaces

Framework 211
This interface acts as a "facade" for the entire Peer2Me framework as the methods in this interface is the only methods the MIDlets running the framework needs access to.

FrameworkListener 214
This interface must be implemented by all Peer2Me MIDlets.

Classes

FrameworkFrontEnd 216
This is the main class of the Peer2Me framework.

C.2.1 Interface Framework

This interface acts as a "facade" for the entire Peer2Me framework as the methods in this interface is the only methods the MIDlets running the framework needs access to. To use the Peer2Me framework, the MIDlets should run the FrameworkFrontEnd.getInstance() which returns a reference of type Framework. All framework services is then available through this reference.

Declaration

public interface Framework

All known subinterfaces

FrameworkFrontEndpeer2me.framework.FrameworkFrontEnd

All classes known to implement interface

FrameworkFrontEndpeer2me.framework.FrameworkFrontEnd

Method summary

connectToNodes(String[]) This method connects multiple devices in a network.
getFileList(String) This method returns a list of the files in the given root directory on the device
initFramework(String, String, String) This method initiates the framework, and is the first method that should be run after getting a instance of the framework.
sendFilePackage(String[], String) This method sends a file package over the network.
sendTextPackage(String[], String) This method sends a text package over the network.
shutdownFramework() This method shuts down the framework and closes all the open network connections and streams.
startNodeSearch() This method starts a search for devices running the same MIDlet.

Methods

* **connectToNodes**
  `void` **connectToNodes**`( java.lang.String[]` **addresses** `)`
  - **Description**
    This method connects multiple devices in a network. When a connection is established, the notifyAboutParticipants() method specified by the FrameworkListener interface is called.
  - **Parameters**
    * `addresses` – The addresses of the devices to connect to.
* **getFileList**
  `java.util.Enumeration` **getFileList**`( java.lang.String` **root** `)`
  - **Description**
    This method returns a list of the files in the given root directory on the device
  - **Parameters**
    * `root` – The path to the root directory
  - **Returns** – An enumeration containing the names of the files in the root directory

* **initFramework**
  `void` **initFramework**`( java.lang.String` **nodeName**`, java.lang.String` **midletName**`, java.lang.String` **preferredNetwork** `) throws java.lang.ClassNotFoundException, java.lang.IllegalAccessException, java.lang.InstantiationException, java.io.IOException, java.lang.Exception`

  - **Description**

    This method initiates the framework, and is the first method that should be run after getting a instance of the framework. It initiates the fundamental services offered by the framework.

  - **Parameters**

    * `nodeName` – The name of the user of the MIDlet.
    * `midletName` – The name of the MIDlet, eventually translated into a ServiceID used to find other devices running the same MIDlet.
    * `preferredNetwork` – Deciding which network implementation to use.

  - **Throws**

    * `java.lang.ClassNotFoundException` – The input preferredNetwork is invalid
    * `java.lang.IllegalAccessException` – The input preferredNetwork is invalid
    * `java.lang.InstantiationException` – The input preferredNetwork is invalid
    * `java.io.IOException` – Error initiating framework
    * `java.lang.Exception` – Error initiating framework

* **sendFilePackage**
  `void` **sendFilePackage**`( java.lang.String[]` **recipients**`, java.lang.String` **filePath** `)`

  - **Description**

    This method sends a file package over the network. When the package terminates to the recipients, they are alerted by the notifyAboutReceivedFilePackage() method specified by the FrameworkListener interface.

  - **Parameters**

    * `recipients` – A list containing the addresses of the recipient nodes
    * `filePath` – The path of the file to be sent

* **sendTextPackage**
  `void` **sendTextPackage**`( java.lang.String[]` **recipients**`, java.lang.String` **textMessage** `)`

  - **Description**

    This method sends a text package over the network. When the package terminates to the recipients, they are alerted by the notifyAboutReceivedTextPackage() method specified by the FrameworkListener interface.

  - **Parameters**

    * `recipients` – A list containing the addresses of the recipient nodes
    * `textMessage` – The text message to be sent

* **shutdownFramework**
  `void` **shutdownFramework**`( )`

  - **Description**

    This method shuts down the framework and closes all the open network connections and streams. It should be called before closing the MIDlet to clean up the network connections.

* **startNodeSearch**
  `void` **startNodeSearch**`( ) throws java.io.IOException`

  - **Description**

    This method starts a search for devices running the same MIDlet. When such a device is found, the notifyAboutFoundNode() method specified by the FrameworkListener interface is called.

  - **Throws**

    * `java.io.IOException` – Thrown if the search crashes

C.2.2 Interface FrameworkListener

This interface must be implemented by all Peer2Me MIDlets. It ensures that the Framework can access a set of methods in the MIDlet in order to notify the MIDlet about various events.

Declaration

public interface FrameworkListener

Method summary

notifyAboutException(String, Exception) This method is called by the framework whenever an exception notice is given by the log.

notifyAboutFoundNode(String, String) This method is called by the framework when a node is found.

notifyAboutParticipants(Hashtable) This method is called from from the framework to notify the midlet about the participants of the ad hoc network.

notifyAboutReceivedFilePackage(String, String) This method is called from the framework whenever a file package is received from a remote node.

notifyAboutReceivedTextPackage(String, String) This method is called from the framework whenever a text package is received from a remote node.

Methods

* **notifyAboutException**
  `void` **notifyAboutException**`( java.lang.String` **location**`, java.lang.Exception` **exception** `)`

- **Description**

  This method is called by the framework whenever an exception notice is given by the log. This will be done in cases where exceptions occure in threads and cannot be thrown in the usual way.

- **Parameters**

  * `location` – The location where the Exception occured
  * `exception` – The actual Exception

* **notifyAboutFoundNode**
  `void` **notifyAboutFoundNode**`( java.lang.String` **nodeAddress**`, java.lang.String` **remoteNodeName** `)`

  - **Description**

    This method is called by the framework when a node is found. These nodes are not yet connected in a network. To do this, use the Framework.connectToNodes() method.

  - **Parameters**

    * `nodeAddress` – The network address of the node
    * `remoteNodeName` – The name of the found remote node

* **notifyAboutParticipants**
  `void` **notifyAboutParticipants**`( java.util.Hashtable` **participants** `)`

  - **Description**

    This method is called from from the framework to notify the midlet about the participants of the ad hoc network.

  - **Parameters**

    * `participants` – A hashtable that contains the names of the participants as unique keys and the network addresses as values.

* **notifyAboutReceivedFilePackage**
  `void` **notifyAboutReceivedFilePackage**`( java.lang.String` **senderName**`, java.lang.String` **filePath** `)`

  - **Description**

    This method is called from the framework whenever a file package is received from a remote node.

  - **Parameters**

    * `senderName` – The name of the sender
    * `filePath` – The path to the received file

* **notifyAboutReceivedTextPackage**
  `void` **notifyAboutReceivedTextPackage**`( java.lang.String` **senderName**`, java.lang.String` **textMessage** `)`

- **Description**

  This method is called from the framework whenever a text package is received from a remote node.

- **Parameters**

  * `senderName` – The name of the sender
  * `textMessage` – The received text message

C.2.3 Class FrameworkFrontEnd

This is the main class of the Peer2Me framework. It manages and connects the resources and functions of the framework. It also handles all communication and interaction with the MIDlets running the framework.

Declaration

public class FrameworkFrontEnd
extends java.lang.Object java.lang.Object
implements Framework

Method summary

connectToNodes(String[]) This method establishes a connection to the chosen nodes.

getFileList(String) This method returns a list of the files in the given root directory on the device

getGroup() This method returns the local representation of the group.

getInstance(FrameworkListener) This method creates an instance of FrameworkFrontEnd and returns it as a reference of type Framework.

getLocalNode() This method returns a reference to the local node.

initFramework(String, String, String) This method initiates the framework, and is the first method that should be run after getting a instance of the framework.

notifyAboutException(String, Exception) This method passes on the Exception notice from the Log to the MIDlet.

notifyAboutFoundNode(String, String) This method is called from the nodeFound() method in the Network class whenever a node is found

notifyAboutLostNode(String) This method removes a lost node from the group.

notifyAboutReceivedFilePackage(FilePackage) This method is called from NodeConnection.processIncomingData() whenever a file package is received from a remote node.

notifyAboutReceivedGroupSyncPackage(GroupSyncPackage) This method is called from NodeConnection.processIncomingData() whenever a groupSyncPackage is received from a remote node.

notifyAboutReceivedTextPackage(TextPackage) This method is called from NodeConnection.processIncomingData() whenever a text package is received from a remote node.

sendFilePackage(String[], String) This method is used by the MIDlet to send a file package over the network.

sendTextPackage(String[], String) This method is used by the MIDlet to send a text package over the network.

shutdownFramework() This method shuts down the framework and closes all the open network connections and streams.

startNodeSearch() This method starts a search for devices running the same MIDlet.

Methods

* **connectToNodes**
  `public void` **connectToNodes**`( java.lang.String[]` **addresses** `)`
  - **Description**
    This method establishes a connection to the chosen nodes. After updating the local group, it synchronizes the groups on all other participating nodes. The method should be called from the MIDlet.
  - **Parameters**
    * `addresses` – The addresses to the nodes to connect to.
* **getFileList**
  `public java.util.Enumeration` **getFileList**`( java.lang.String` **root** `)`
  - **Description**
    This method returns a list of the files in the given root directory on the device
  - **Parameters**
    * `root` – The path to the root directory
  - **Returns** – A Enumeration containing the names of the files in the root directory
* **getGroup**
  `public peer2me.domain.Group` **getGroup**`( )`
  - **Description**
    This method returns the local representation of the group. It is called from ConnectionListener.run() or Network.nodeFound() when a remote node is found and should be added to the group.
  - **Returns** – The local representation of the group
* **getInstance**
  `public static synchronized Framework` **getInstance**`( FrameworkListener` **midlet** `)`
  - **Description**
    This method creates an instance of FrameworkFrontEnd and returns it as a reference of type Framework. This is the only method that can be called directly from the MIDlet on the

FrameworkFrontEnd. The MIDlet is restricted to only use the methods specified in the Framework interface.

- **Parameters**
  - `midlet` – A reference to the MIDlet (The MIDlet must implement the FrameworkListener interface).
- **Returns** – A reference to the Framework

* **getLocalNode**
  `public peer2me.domain.Node` **getLocalNode**`( )`
  - **Description**
    This method returns a reference to the local node.
  - **Returns** – An object representing the local node

* **initFramework**
  `public void` **initFramework**`( java.lang.String` **nodeName**`, java.lang.String` **midletName**`, java.lang.String` **preferredNetwork** `) throws java.lang.ClassNotFoundException, java.lang.IllegalAccessException, java.lang.InstantiationException, java.io.IOException, java.lang.Exception`
  - **Description**
    This method initiates the framework, and is the first method that should be run after getting a instance of the framework. It initiates the fundamental services offered by the framework.
  - **Parameters**
    - `nodeName` – The name of the user of the MIDlet.
    - `midletName` – The name of the MIDlet, eventually translated into a ServiceID used to find other devices running the same MIDlet.
    - `preferredNetwork` – Deciding which network implementation to use.
  - **Throws**
    - `java.lang.ClassNotFoundException` – The input preferredNetwork is invalid
    - `java.lang.IllegalAccessException` – The input preferredNetwork is invalid
    - `java.lang.InstantiationException` – The input preferredNetwork is invalid
    - `java.io.IOException` – Error initiating framework
    - `java.lang.Exception` – Error initiating framework

* **notifyAboutException**
  `public void` **notifyAboutException**`( java.lang.String` **location**`, java.lang.Exception` **exception** `)`
  - **Description**
    This method passes on the Exception notice from the Log to the MIDlet. This will be done in cases where exceptions occure in threads and cannot be thrown in the usual way.

- **Parameters**
  - `location` – The location (class and method) where the Exception occured
  - `exception` – The actual Exception

* **notifyAboutFoundNode**
  `public void` **notifyAboutFoundNode**`( java.lang.String` **address**`, java.lang.String` **remoteNodeName** `)`

  - **Description**

    This method is called from the nodeFound() method in the Network class whenever a node is found
  - **Parameters**
    - `address` – The network address of the node
    - `remoteNodeName` – The name of the found remote node

* **notifyAboutLostNode**
  `public synchronized void` **notifyAboutLostNode**`( java.lang.String` **address** `)`

  - **Description**

    This method removes a lost node from the group. It is called from Network.sendDataPackage() if a node is unreachable. After removing the node, the groups on all other nodes become synchronized.
  - **Parameters**
    - `address` – The address to the lost node

* **notifyAboutReceivedFilePackage**
  `public void` **notifyAboutReceivedFilePackage**`( peer2me.domain.FilePackage` **filePackage** `)`

  - **Description**

    This method is called from NodeConnection.processIncomingData() whenever a file package is received from a remote node. It processes the package, logs the event, and notifies the midlet.
  - **Parameters**
    - `filePackage` – The received file package.

* **notifyAboutReceivedGroupSyncPackage**
  `public void` **notifyAboutReceivedGroupSyncPackage**`( peer2me.domain.GroupSyncPackage` **groupSyncPackage** `)`

  - **Description**

    This method is called from NodeConnection.processIncomingData() whenever a groupSyncPackage is received from a remote node. The method processes the package, logs the event, and updates the group.
  - **Parameters**

    * `groupSyncPackage` – The received groupSyncPackage.

* **notifyAboutReceivedTextPackage**
  `public void` **notifyAboutReceivedTextPackage**`( peer2me.domain.TextPackage` **textPackage** `)`

  - **Description**

    This method is called from NodeConnection.processIncomingData() whenever a text package is received from a remote node. It processes the package, logs the event, and notifies the midlet.

  - **Parameters**

    * `textPackage` – The received text package.

* **sendFilePackage**
  `public void` **sendFilePackage**`( java.lang.String[]` **recipients**`, java.lang.String` **filePath** `)`

  - **Description**

    This method is used by the MIDlet to send a file package over the network. When the package terminates to the recipients, the notifyAboutReceivedFilePackage() method in this class is run.

  - **Parameters**

    * `recipients` – A list containing the addresses of the recipient nodes
    * `filePath` – The path of the file to send

* **sendTextPackage**
  `public void` **sendTextPackage**`( java.lang.String[]` **recipients**`, java.lang.String` **textMessage** `)`

  - **Description**

    This method is used by the MIDlet to send a text package over the network. When the package terminates to the recipients, the notifyAboutReceivedTextPackage() method in this class is run.

  - **Parameters**

    * `recipients` – A list containing the addresses of the recipient nodes
    * `textMessage` – The text to be sent

* **shutdownFramework**
  `public void` **shutdownFramework**`( )`

  - **Description**

    This method shuts down the framework and closes all the open network connections and streams. It should be called from the MIDlet before closing, to clean up the network connections.

* **startNodeSearch**
  `public void` **startNodeSearch**`( ) throws java.io.IOException`

- **Description**

  This method starts a search for devices running the same MIDlet. When such a device is found, the notifyAboutFoundNode() method in this class is called.

- **Throws**

  * `java.io.IOException` – Thrown if the search crashes

C.3 Package peer2me.network.bluetooth

Package Contents

Interfaces

BluetoothServiceDiscoveryListener 221
This interface has to be implemented by classes that wants to do a Bluetooth service discovery using the BluetoothServiceDiscovery class, and receive callbacks from this class.

Classes

BluetoothNetwork 222
This class is a bluetooth specific sub class of the Network class and implements all the abstract methods of it's parent class in a bluetooth context.
BluetoothServiceDiscovery 225
This class is responsible for doing the low level Bluetooth discovery operations.

C.3.1 Interface BluetoothServiceDiscoveryListener

This interface has to be implemented by classes that wants to do a Bluetooth service discovery using the BluetoothServiceDiscovery class, and receive callbacks from this class. In this case, the class BluetoothNetwork implements this interface.

Declaration

public interface BluetoothServiceDiscoveryListener

All known subinterfaces

BluetoothNetworkpeer2me.network.bluetooth.BluetoothNetwork

All classes known to implement interface

BluetoothNetworkpeer2me.network.bluetooth.BluetoothNetwork

Method summary

serviceDiscoveryError() What to do when something went wrong during servicediscovery
serviceSearchCompleted() What to do when serviceSearch is completed

Methods

* **serviceDiscoveryError**
  `void` **serviceDiscoveryError**`( )`
  - **Description**
    What to do when something went wrong during servicediscovery
* **serviceSearchCompleted**
  `void` **serviceSearchCompleted**`( )`
  - **Description**
    What to do when serviceSearch is completed

C.3.2 Class BluetoothNetwork

This class is a bluetooth specific sub class of the Network class and implements all the abstract methods of it's parent class in a bluetooth context. It uses the bluetooth Java API, JSR-82, to perform operations on the bluetooth hardware of the mobile device.

Declaration

public class BluetoothNetwork
extends peer2me.network.Network peer2me.network.Network
implements BluetoothServiceDiscoveryListener

Constructor summary

BluetoothNetwork() Constructor.

Method summary

connectionEstablished() This method is called from the ConnectionListener.run() when the acceptAndOpen() method in ConnectionListener.run() is done.
connectToNode(String) This method establishes a connection to the chosen node.
getNodeAddress(Object) This method returns the node address.
getRemoteNodeName(Object) This method fetches the name of the remote node.
getUUIDString() This method returns the UUID string used as an identifier in the discovery process.
init() Initiates the network instance.
nodeFound(Object) Called when the same MIDlet is found on a remote device.
searchForNodes() Starts a search for devices running the same MIDlet
sendDataPackage(DataPackage, String[]) This method is used by the FrameworkFrontEnd to send a data package of any sort to a remote node.
serviceDiscoveryError() What to do when something went wrong during servicediscovery.
serviceSearchCompleted() Sets the boolean serviceSearchCompleted = true.

Constructors

* **BluetoothNetwork**
  `public` **BluetoothNetwork**`( )`

  - **Description**

    Constructor. Protected to ensure singleton pattern.

Methods

* **connectionEstablished**
  `public void` **connectionEstablished**`( )`

  - **Description**

    This method is called from the ConnectionListener.run() when the acceptAndOpen() method in ConnectionListener.run() is done.

* **connectToNode**
  `public void` **connectToNode**`( java.lang.String` **nodeAddress** `)`

  - **Description**

    This method establishes a connection to the chosen node. It is run from the BluetoothNetwork.sendDataPackage().

  - **Parameters**

    * `nodeAddress` – The address to the node to connect to

* **getNodeAddress**
  `public java.lang.String` **getNodeAddress**`( java.lang.Object` **input** `) throws java.io.IOException`

  - **Description**

    This method returns the node address.

  - **Parameters**

    * `input` – String "localNode" to retreive the address of the local device. A ServiceRecord or StreamConnection object to retreive the address of a remote device.

  - **Returns** – The node network address.

  - **Throws**

    * `java.io.IOException` –

* **getRemoteNodeName**
  `public java.lang.String` **getRemoteNodeName**`( java.lang.Object` **input** `)`

  - **Description**

    This method fetches the name of the remote node.

- **Parameters**
  - `input` – An object representing the connection to the found node.
- **Returns** – The name of the remote node.

* **getUUIDString**
  `public java.lang.String` **getUUIDString**`( )`

  - **Description**
    This method returns the UUID string used as an identifier in the discovery process. The UUID string is generated based on the application ID given by the application running the framework. The UUID must be used to ensure that all nodes joining the network are running the same application.
  - **Returns** – uuidString

* **init**
  `public void` **init**`( ) throws javax.bluetooth.BluetoothStateException`

  - **Description**
    Initiates the network instance. It is called from the FrameworkFrontEnd.initFramework()
  - **Throws**
    - `javax.bluetooth.BluetoothStateException` – Failed to initiate the network

* **nodeFound**
  `public void` **nodeFound**`( java.lang.Object` **input** `)`

  - **Description**
    Called when the same MIDlet is found on a remote device. It is called from BluetoothServiceDiscovery.serviceSearchCompleted().
  - **Parameters**
    - `input` – Either a ServiceRecord or a StreamConnection that describes the characteristics of the Bluetooth service found

* **searchForNodes**
  `public void` **searchForNodes**`( ) throws java.io.IOException`

  - **Description**
    Starts a search for devices running the same MIDlet
  - **Throws**
    - `java.io.IOException` – Error during the search

* **sendDataPackage**
  `public void` **sendDataPackage**`( peer2me.domain.DataPackage` **dataPackage**`, java.lang.String[]` **recipients** `)`

- **Description**

  This method is used by the FrameworkFrontEnd to send a data package of any sort to a remote node.

- **Parameters**

  * `dataPackage` – The data package to be sent
  * `recipients` – A list containing addresses to the recipient nodes

* **serviceDiscoveryError**
  `public void` **serviceDiscoveryError**`( )`

  - **Description**

    What to do when something went wrong during servicediscovery. The method is called from BluetoothServiceDiscovery.serviceSearchCompleted().

* **serviceSearchCompleted**
  `public void` **serviceSearchCompleted**`( )`

  - **Description**

    Sets the boolean serviceSearchCompleted = true. This value will interrupt the while-loop in sendDataPackage. This because the serviceSearch must be completed before we try to send a package. The method is called from BluetoothServiceDiscovery.serviceSearchCompleted().

Members inherited from class `peer2me.network.Network` peer2me.network.Network

* `public abstract void` **connectionEstablished**`( )`
* `public abstract void` **connectToNode**`( java.lang.String` **nodeAddress** `)`
* `public String` **getApplicationId**`( )`
* `public ConnectionListener` **getConnectionListener**`( )`
* `public FrameworkFrontEnd` **getFrameworkFrontEnd**`( )`
* `public static synchronized Network` **getInstance**`( )`
* `public static synchronized Network` **getInstance**`( java.lang.String` **preferredNetwork** `)`
  `throws java.lang.ClassNotFoundException, java.lang.IllegalAccessException, java.lang.InstantiationException`
* `public abstract String` **getNodeAddress**`( java.lang.Object` **input** `) throws java.io.IOException`
* `public abstract String` **getRemoteNodeName**`( java.lang.Object` **input** `)`
* `public abstract void` **init**`( ) throws java.lang.Exception`
* `public abstract void` **nodeFound**`( java.lang.Object` **input** `) throws java.io.IOException`
* `public abstract void` **searchForNodes**`( ) throws java.io.IOException`
* `public abstract void` **sendDataPackage**`( peer2me.domain.DataPackage` **dataPackage**`, java.lang.String[]` **recipients** `)`
* `public void` **setApplicationId**`( java.lang.String` **applicationID** `)`
* `public void` **setConnectionListener**`( ConnectionListener` **connectionListener** `)`
* `public void` **setFrameworkFrontEnd**`( peer2me.framework.FrameworkFrontEnd` **frameworkFrontEnd** `)`

C.3.3 Class BluetoothServiceDiscovery

This class is responsible for doing the low level Bluetooth discovery operations. The class initializes seqential device discovery, and searches for services (the same MIDlet built upon the Peer2Me framework) on each of the found devices.

Declaration

public class BluetoothServiceDiscovery
extends java.lang.Object java.lang.Object
implements javax.bluetooth.DiscoveryListener

Constructor summary

BluetoothServiceDiscovery() Constructor.

Method summary

deviceDiscovered(RemoteDevice, DeviceClass) This method is called by the javax.bluetooth.DiscoveryAgent (agent) whenever a bluetooth device is discovered
doDeviceDiscovery() This method starts the discovery process.
inquiryCompleted(int) This method is called by the javax.bluetooth.DiscoveryAgent (agent) when the discovery process is completed
servicesDiscovered(int, ServiceRecord[]) This method is called by the javax.bluetooth.DiscoveryAgent (agent) whenever one or more services (read: Peer2Me framework) are found on a remote device
serviceSearchCompleted(int, int) This method is called by the javax.bluetooth.DiscoveryAgent (agent) when the search for services (read: Peer2Me framework) is completed
startServiceSearch(String) This method is used to re-establish a connection to a device when we have the address.

Constructors

* **BluetoothServiceDiscovery**
  `public` **BluetoothServiceDiscovery**`( )`
  - **Description**
    Constructor. Called from BluetoothNetwork.init().

Methods

* **deviceDiscovered**
  `public void` **deviceDiscovered**`( javax.bluetooth.RemoteDevice` **remoteDevice**`, javax.bluetooth.DeviceClass` **deviceClass** `)`
  - **Description**
    This method is called by the javax.bluetooth.DiscoveryAgent (agent) whenever a bluetooth device is discovered
  - **Parameters**
    * `remoteDevice` – The device discovered
    * `deviceClass` – The device class of the discovered device

* **doDeviceDiscovery**
  `public void` **doDeviceDiscovery**`( ) throws javax.bluetooth.BluetoothStateException`
  - **Description**
    This method starts the discovery process. It is called from BluetoothNetwork.searchForNodes().
  - **Throws**
    * `javax.bluetooth.BluetoothStateException` – Error getting reference to LocalDevice
* **inquiryCompleted**
  `public void` **inquiryCompleted**`( int` **discType** `)`
  - **Description**
    This method is called by the javax.bluetooth.DiscoveryAgent (agent) when the discovery process is completed
  - **Parameters**
    * `discType` – The type of request that was completed; either INQUIRY_COMPLETED, INQUIRY_TERMINATED, or INQUIRY_ERROR
* **servicesDiscovered**
  `public void` **servicesDiscovered**`( int` **transId**`, javax.bluetooth.ServiceRecord[]` **serviceRecord** `)`
  - **Description**
    This method is called by the javax.bluetooth.DiscoveryAgent (agent) whenever one or more services (read: Peer2Me framework) are found on a remote device
  - **Parameters**
    * `transId` – The transaction ID of the service search that is posting the result
    * `serviceRecord` – A list of services found during the search request
* **serviceSearchCompleted**
  `public void` **serviceSearchCompleted**`( int` **transID**`, int` **respCode** `)`
  - **Description**
    This method is called by the javax.bluetooth.DiscoveryAgent (agent) when the search for services (read: Peer2Me framework) is completed
  - **Parameters**
    * `transID` – The transaction ID of the service search that is posting the result
    * `respCode` – The response code that indicates the status of the transaction
* **startServiceSearch**
  `public void` **startServiceSearch**`( java.lang.String` **address** `)`
  - **Description**
    This method is used to re-establish a connection to a device when we have the address.
  - **Parameters**
    * `address` – The address to the device

C.4 Package peer2me.network

Package Contents

Classes

ConnectionListener 228
This class contains a ConnectionListener thread listening for incoming connection attempts from other devices running the same MIDlet built upon the framework.

Network 229
This is the super class of the technology specific network classes.

NodeConnection 233
This class contains a thread that runs on each connected node and listens for incoming data packages and sends data packages out.

C.4.1 Class ConnectionListener

This class contains a ConnectionListener thread listening for incoming connection attempts from other devices running the same MIDlet built upon the framework. When a incomming connection is detected, a Node representation is created representing the connecting device. A ConnectionListener thread is created in Network.init().

Declaration

public class ConnectionListener
extends java.lang.Object java.lang.Object
implements java.lang.Runnable

Constructor summary

ConnectionListener(String) Constructor.

Method summary

run() This method is called when the ConnectionListener thread is started in the constructor.
shutdown() This method shuts down this thread and closes the connection to clean up.

Constructors

* **ConnectionListener**
  `public` **ConnectionListener**`( java.lang.String` **connectionURL** `)`
  - **Description**
    Constructor. A ConnectionListener is created in the Network.init() method.
  - **Parameters**
    * `connectionURL` – The ConnectionURL to listen to

Methods

* **run**
  `public void run( )`
  - **Description**
    This method is called when the ConnectionListener thread is started in the constructor. It continously listens for incoming connections matching the serviceID of the peer2me framework. The listener is "passive" and opens a connection waiting for a device to take contact. If an incoming connetion occurs, information is abstracted from the remote node, and a node object containing this connection is created and added to the group on the local node.

* **shutdown**
  `public void shutdown( )`
  - **Description**
    This method shuts down this thread and closes the connection to clean up. It is called from FrameworkFrontEnd.shutdownFramework().

C.4.2 Class Network

This is the super class of the technology specific network classes. Methods that are equal for all the sub classes are located in this super class, and there are abstact methods that the sub classes have to implement. The getInstance() method in this class returns a reference to the preferred network sub class.

Declaration

public abstract class Network
extends java.lang.Object java.lang.Object

All known subclasses

BluetoothNetworkpeer2me.network.bluetooth.BluetoothNetwork

Constructor summary

Network()

Method summary

connectionEstablished() This method is called from the ConnectionListener.run() when the acceptAndOpen() method in ConnectionListener.run() is done.
connectToNode(String) This method establishes a connection to the chosen node.
getApplicationId() This method returns the applicationId
getConnectionListener() This method returns the ConnectionListener reference
getFrameworkFrontEnd() This method returns the FrameworkFrontEnd reference

getInstance() This method returns a reference to the instance of the preferred network.

getInstance(String) This method returns an instance of the preferred network.

getNodeAddress(Object) This method returns the node address.

getRemoteNodeName(Object) This method fetches the name of the remote node.

init() Initiates the network instance.

nodeFound(Object) Called when the same MIDlet is found on a remote device

searchForNodes() Starts a search for devices running the same MIDlet

sendDataPackage(DataPackage, String[]) This method is used by the FrameworkFrontEnd to send a data package of any sort to a remote node.

setApplicationId(String) This method sets the applicationID.

setConnectionListener(ConnectionListener) This method sets a reference to the ConnectionListener

setFrameworkFrontEnd(FrameworkFrontEnd) This method sets a reference to the FrameworkFrontEnd

Constructors

* **Network**
  `public` **Network**`( )`

Methods

* **connectionEstablished**
  `public abstract void` **connectionEstablished**`( )`

  - **Description**
    This method is called from the ConnectionListener.run() when the acceptAndOpen() method in ConnectionListener.run() is done.

* **connectToNode**
  `public abstract void` **connectToNode**`( java.lang.String` **nodeAddress** `)`

  - **Description**
    This method establishes a connection to the chosen node. It could e.g. be run from the Network.sendDataPackage() to connect before sending a package.
  - **Parameters**
    * `nodeAddress` – The address to the node to connect to

* **getApplicationId**
  `public java.lang.String` **getApplicationId**`( )`

  - **Description**
    This method returns the applicationId
  - **Returns** – applicationID The ID of the MIDlet

* **getConnectionListener**
  `public ConnectionListener` **getConnectionListener**`( )`

  - **Description**

    This method returns the ConnectionListener reference

  - **Returns** – connectionListener A reference to the ConnectionListener

* **getFrameworkFrontEnd**
  `public peer2me.framework.FrameworkFrontEnd` **getFrameworkFrontEnd**`( )`

  - **Description**

    This method returns the FrameworkFrontEnd reference

  - **Returns** – frameworkFrontEnd A reference to the FrameworkFrontEnd

* **getInstance**
  `public static synchronized Network` **getInstance**`( )`

  - **Description**

    This method returns a reference to the instance of the preferred network. It is used when an instance already is created and a reference to this instance is needed.

  - **Returns** – The Network instance

  - **Throws**

    * `java.lang.ClassNotFoundException` – The input preferredNetwork is invalid
    * `java.lang.IllegalAccessException` – The input preferredNetwork is invalid
    * `java.lang.InstantiationException` – The input preferredNetwork is invalid

* **getInstance**
  `public static synchronized Network` **getInstance**`( java.lang.String` **preferredNetwork** `) throws java.lang.ClassNotFoundException, java.lang.IllegalAccessException, java.lang.InstantiationException`

  - **Description**

    This method returns an instance of the preferred network. It is called from FrameworkFrontEnd.initFramework().

  - **Parameters**

    * `preferredNetwork` – Indicating which network implementation to use. *

  - **Returns** – The Network instance

  - **Throws**

    * `java.lang.ClassNotFoundException` – The input preferredNetwork is invalid
    * `java.lang.IllegalAccessException` – The input preferredNetwork is invalid
    * `java.lang.InstantiationException` – The input preferredNetwork is invalid

* **getNodeAddress**
  `public abstract java.lang.String` **getNodeAddress**`( java.lang.Object` **input** `)`
  `throws java.io.IOException`

  - **Description**
    This method returns the node address.
  - **Parameters**
    * `input` – String "localNode" to retreive the address of the local device. A object representing the connection to the remote node to retreive the address of a remote device.
  - **Returns** – The node network address.
  - **Throws**
    * `java.io.IOException` –

* **getRemoteNodeName**
  `public abstract java.lang.String` **getRemoteNodeName**`( java.lang.Object` **input** `)`

  - **Description**
    This method fetches the name of the remote node.
  - **Parameters**
    * `input` – An object representing the connection to the found node.
  - **Returns** – The name of the remote node.

* **init**
  `public abstract void` **init**`( ) throws java.lang.Exception`

  - **Description**
    Initiates the network instance. It is called from the FrameworkFrontEnd.initFramework()
  - **Throws**
    * `java.lang.Exception` – Failed to initiate the network

* **nodeFound**
  `public abstract void` **nodeFound**`( java.lang.Object` **input** `) throws java.io.IOException`

  - **Description**
    Called when the same MIDlet is found on a remote device
  - **Parameters**
    * `input` – An object representing the connection to the found node.

* **searchForNodes**
  `public abstract void` **searchForNodes**`( ) throws java.io.IOException`

  - **Description**
    Starts a search for devices running the same MIDlet

- **Throws**
  - `java.io.IOException` – Error during the search

* **sendDataPackage**
  `public abstract void` **sendDataPackage**`( peer2me.domain.DataPackage` **dataPackage**`, java.lang.String[]` **recipients** `)`
  - **Description**
    This method is used by the FrameworkFrontEnd to send a data package of any sort to a remote node.
  - **Parameters**
    - `dataPackage` – The data package to be sent
    - `recipients` – A list containing addresses to the recipient nodes

* **setApplicationId**
  `public void` **setApplicationId**`( java.lang.String` **applicationID** `)`
  - **Description**
    This method sets the applicationID. The application ID must be used to ensure that all nodes joining the network are running the same MIDlet.
  - **Parameters**
    - `applicationID` – The ID of the MIDlet (e.g. the MIDlet name)

* **setConnectionListener**
  `public void` **setConnectionListener**`( ConnectionListener` **connectionListener** `)`
  - **Description**
    This method sets a reference to the ConnectionListener
  - **Parameters**
    - `connectionListener` – A reference to the ConnectionListener

* **setFrameworkFrontEnd**
  `public void` **setFrameworkFrontEnd**`( peer2me.framework.FrameworkFrontEnd` **frameworkFrontEnd** `)`
  - **Description**
    This method sets a reference to the FrameworkFrontEnd
  - **Parameters**
    - `frameworkFrontEnd` – A reference to the FrameworkFrontEnd

C.4.3 Class NodeConnection

This class contains a thread that runs on each connected node and listens for incoming data packages and sends data packages out. It is created and started in NodeConnection.startNodeConnection().

Declaration

public class NodeConnection
extends java.lang.Object java.lang.Object

Constructor summary

NodeConnection(StreamConnection, Node) Constructor.

Method summary

closeConnection() This method closes the input- and output streams and the connection.
getConnection() This method returns the connection object.
getSendQueueSize() This method return the size of the sendQue.
openInputStream() This method sets a boolean that controls whether or not the InputStream are allowed to listen for incoming data.
openOutputStream() This method sets a boolean that controls whether or not the OutputStream are allowed to send data.
processIncomingData() This method receives incoming datapackages from remote nodes.
processSendQueue() This method sends datapackages to remote nodes.
sendDataPackage(DataPackage) This method is called by the sendMessage() method in the Network class when a data package is sent to the Node associated with this NodeConnection.
setConnection(StreamConnection) This method updates the connection object.

Constructors

* **NodeConnection**
  `public` **NodeConnection**`( javax.microedition.io.StreamConnection` **connection**`, peer2me.domain.Node` **node** `)`
  - **Description**
    Constructor. This constructor is called from the constructor in class Node.
  - **Parameters**
    * `connection` – The connection to the node
    * `node` – The node that owns this NodeConnection

Methods

* **closeConnection**
  `public void` **closeConnection**`( )`
  - **Description**
    This method closes the input- and output streams and the connection. It is called from Group.shutdownGroup() to clean up during shutdown.

* **getConnection**
  `public javax.microedition.io.StreamConnection` **getConnection**`( )`
  - **Description**
    This method returns the connection object.
  - **Returns** – An object representing the connection to the remote node
* **getSendQueueSize**
  `public int` **getSendQueueSize**`( )`
  - **Description**
    This method return the size of the sendQue.
  - **Returns** – The size of the sendQue
* **openInputStream**
  `public void` **openInputStream**`( )`
  - **Description**
    This method sets a boolean that controls whether or not the InputStream are allowed to listen for incoming data. The value is toggled from ConnectionListener.run().
* **openOutputStream**
  `public void` **openOutputStream**`( )`
  - **Description**
    This method sets a boolean that controls whether or not the OutputStream are allowed to send data. The value is toggled from NodeConnection.sendDataPackage()
* **processIncomingData**
  `public void` **processIncomingData**`( )`
  - **Description**
    This method receives incoming datapackages from remote nodes. It is called in an infinite loop in the private class InputThread in this class.
* **processSendQueue**
  `public synchronized void` **processSendQueue**`( )`
  - **Description**
    This method sends datapackages to remote nodes. It processes the que of unsent datapackages. It is called in an infinite loop in the private class OutputThread in this class.
* **sendDataPackage**
  `public synchronized void` **sendDataPackage**`( peer2me.domain.DataPackage` **dataPackage** `)`
  - **Description**
    This method is called by the sendMessage() method in the Network class when a data package is sent to the Node associated with this NodeConnection.

- **Parameters**
  - `dataPackage` – The DataPackage to send

* **setConnection**
  `public void` **setConnection**`( javax.microedition.io.StreamConnection` **connection** `)`

  - **Description**

    This method updates the connection object. It is used when the existing connection is closed and a new open connection is needed.

  - **Parameters**
    - `connection` – The connection to the remote node

C.5 Package peer2me.util

Package Contents

Classes

ASCIIToHexConvert 236
This class converts ASCII letters into hexadecimal numbers

FileHandler 237
This class contains functionality for reading and writing all kinds of files to and from the device file system.

Log 239
This class contains functionality to create and maintain a log of events and exceptions.

C.5.1 Class ASCIIToHexConvert

This class converts ASCII letters into hexadecimal numbers

Declaration

public class ASCIIToHexConvert
extends java.lang.Object java.lang.Object

Constructor summary

ASCIIToHexConvert() Constructor

Method summary

convertASCIIToHex(String) This method returns a String with hex representations of each character in the provided String

Constructors

* **ASCIIToHexConvert**
  `public` **ASCIIToHexConvert**`( )`

  - **Description**

    Constructor

Methods

* **convertASCIIToHex**
  `public java.lang.String` **convertASCIIToHex**`( java.lang.String` **ascii** `)`

  - **Description**

    This method returns a String with hex representations of each character in the provided String

  - **Parameters**

    * `ascii` – The String to convert

  - **Returns** – A String with hex representations

C.5.2 Class FileHandler

This class contains functionality for reading and writing all kinds of files to and from the device file system.

Declaration

public class FileHandler
extends java.lang.Object java.lang.Object

Constructor summary

FileHandler(String) Constructor.

Method summary

closeFile() This method closes and nullifies the input- and ouput streams, and the file connection
getBlockSize() This method fetches the size of the blocks to read and write
getFileList() This method returns a list of the files in the given file path on the device
getFileSize() This method returns the size of the this file
readFile() This method reads the next byte in the file and returns it
setFileSize(long) This method sets the size of the this file
writeFile(byte) This method writes the incoming byte to the file
writeFile(byte[], int) This method writes the incoming byte block to the file

Constructors

* **FileHandler**
  `public` **FileHandler**`( java.lang.String` **filePath** `)`
  - **Description**
    Constructor.
  - **Parameters**
    * `filePath` – The path to the file to be handled

Methods

* **closeFile**
  `public void` **closeFile**`( )`
  - **Description**
    This method closes and nullifies the input- and ouput streams, and the file connection
* **getBlockSize**
  `public int` **getBlockSize**`( )`
  - **Description**
    This method fetches the size of the blocks to read and write
  - **Returns** – The blocksize
* **getFileList**
  `public java.util.Enumeration` **getFileList**`( )`
  - **Description**
    This method returns a list of the files in the given file path on the device
  - **Returns** – A Enumeration containing the names of the files in the root directory
* **getFileSize**
  `public long` **getFileSize**`( )`
  - **Description**
    This method returns the size of the this file
  - **Returns** – The size as a long
* **readFile**
  `public synchronized byte[]` **readFile**`( ) throws java.io.IOException`
  - **Description**
    This method reads the next byte in the file and returns it
  - **Returns** – The next block of bytes

- **Throws**
  - `java.io.IOException` – This exception is thrown when the reading has failed

* **setFileSize**
  `public void` **setFileSize**`( long` **fileSize** `)`
  - **Description**
    This method sets the size of the this file
  - **Parameters**
    - `fileSize` – The size to set

* **writeFile**
  `public synchronized void` **writeFile**`( byte` **theByte** `) throws java.io.IOException`
  - **Description**
    This method writes the incoming byte to the file
  - **Parameters**
    - `theByte` – The next byte to write
  - **Throws**
    - `java.io.IOException` – This exception is thrown when the writing has failed

* **writeFile**
  `public synchronized void` **writeFile**`( byte[]` **theBytes**`, int` **numberOfBytesRead** `) throws java.io.IOException`
  - **Description**
    This method writes the incoming byte block to the file
  - **Parameters**
    - `theBytes` – The next byte block to write
    - `numberOfBytesRead` – The number of bytes in the theBytes[] array
  - **Throws**
    - `java.io.IOException` – This exception is thrown when the writing has failed

C.5.3 Class Log

This class contains functionality to create and maintain a log of events and exceptions. The Log contains four differnet kinds of logs, an exception log, a connection log, a data package log and a debug log. They can be used to log events from anywhere in the framework, and the logs can be retreived later to get information about the execution of the MIDlet.

Declaration

public class Log
extends java.lang.Object java.lang.Object

Field summary

CONNECTION_LOG
DATA_PACKAGE_LOG
DEBUG_LOG
EXCEPTION_LOG

Method summary

getInstance() This method returns the only existing instance of the Log class
getLog(int) This method returns the desired log in a displayable format.
logConnection(String) This method adds a Connection entry to the Connection log
logDataPackage(String) This method adds a data package entry to the data package log
logDebugInfo(String, String) This method adds a Debug entry to the Debug log
logException(String, Exception, boolean) This method adds an Exception entry to the Exception log
setFramework(FrameworkFrontEnd) This method is called by the FrameworkFrontEnd to reveal itself to the Log

Fields

* public static final int **EXCEPTION_LOG**
* public static final int **CONNECTION_LOG**
* public static final int **DATA_PACKAGE_LOG**
* public static final int **DEBUG_LOG**

Methods

* **getInstance**
  `public static synchronized Log` **getInstance**`( )`
  - **Description**
    This method returns the only existing instance of the Log class
  - **Returns** – The singleton instance of the Log class
* **getLog**
  `public java.lang.String` **getLog**`( int` **log** `)`
  - **Description**
    This method returns the desired log in a displayable format. It must be called by activating a soft button in the application. Due to multithreading and interuptions there can be some delay in the creation of the log.
  - **Parameters**

    * `log` – The Log.FIELD representing the desired log
  - **Returns** – The desired log as a String

* **logConnection**
  `public void` **logConnection**`( java.lang.String` **connectionStatus** `)`
  - **Description**
    This method adds a Connection entry to the Connection log
  - **Parameters**
    * `connectionStatus` – A textual description of the connection status

* **logDataPackage**
  `public void` **logDataPackage**`( java.lang.String` **packageStatus** `)`
  - **Description**
    This method adds a data package entry to the data package log
  - **Parameters**
    * `packageStatus` – A textual description of the data package status

* **logDebugInfo**
  `public void` **logDebugInfo**`( java.lang.String` **location**`, java.lang.String` **debugInfo** `)`
  - **Description**
    This method adds a Debug entry to the Debug log
  - **Parameters**
    * `location` – The location (class and method) where the debuginfo was logged
    * `debugInfo` – A textual description of the debug information

* **logException**
  `public void` **logException**`( java.lang.String` **location**`, java.lang.Exception` **exception**`, boolean` **notify** `)`
  - **Description**
    This method adds an Exception entry to the Exception log
  - **Parameters**
    * `location` – The location (class and method) where the Exception occured
    * `exception` – The actual Exception
    * `notify` – This boolean decides whether or not to notify the Framework about the Exception that occured

* **setFramework**
  `public void` **setFramework**`( peer2me.framework.FrameworkFrontEnd` **framework** `)`

- **Description**

  This method is called by the FrameworkFrontEnd to reveal itself to the Log

- **Parameters**

  * `framework` – The FrameworkFrontEnd refrence sent by the FrameworkFrontEnd itself

APPENDIX D

Peer2Me v2.0 Source code

The full source code of Peer2Me v2.0.

D.1 Package peer2me.framework

D.1.1 Interface Framework

```
package peer2me.framework;

import java.io.IOException;
import java.util.Enumeration;

/**
 *
 * This interface acts as a "facade" for the entire Peer2Me framework as the
 * methods in this interface is the only methods the MIDlets running the
 * framework needs access to. To use the Peer2Me framework, the MIDlets should
 * run the FrameworkFrontEnd.getInstance() which returns a
 * reference of type Framework. All framework services is then available
 * through this reference.
 *
 * @author Torbjørn Vatn & Steinar A. Hestnes
 */
public interface Framework{

  /**
   *
```

```
21     * This method initiates the framework, and is the first method that should
22     * be run after getting a instance of the framework. It initiates the
23     * fundamental services offered by the framework.
24     *
25     * @param nodeName The name of the user of the MIDlet.
26     * @param midletName The name of the MIDlet, eventually translated into a ServiceID
27       * used to find other devices running the same MIDlet.
28     * @param preferredNetwork Deciding which network implementation to use.
29     *
30     * @throws ClassNotFoundException The input preferredNetwork is invalid
31     * @throws IllegalAccessException The input preferredNetwork is invalid
32     * @throws InstantiationException The input preferredNetwork is invalid
33     * @throws IOException Error initiating framework
34     * @throws Exception Error initiating framework
35     */
36    public void initFramework(String nodeName, String midletName, String preferredNetwork) throws ClassNotFoundException,
37                                                IllegalAccessException, InstantiationException, IOException, Exception;
38
39
40    /**
41     *
42     * This method shuts down the framework and closes all the open network connections and streams.
43     * It should be called before closing the MIDlet to clean up the network connections.
44     *
45     */
46    public void shutdownFramework();
47
48
49    /**
50     *
51     * This method starts a search for devices running the same MIDlet.
52     * When such a device is found, the notifyAboutFoundNode() method
53     * specified by the FrameworkListener interface is called.
54     *
55     * @throws IOException Thrown if the search crashes
56     */
57    public void startNodeSearch() throws IOException;
58
59
60      /**
61       *
62       * This method connects multiple devices in a network.
63       * When a connection is established, the notifyAboutParticipants()
64       * method specified by the FrameworkListener interface is called.
65       *
```

```
66      * @param addresses The addresses of the devices to connect to.
67      */
68     public void connectToNodes(String[] addresses);
69
70
71     /**
72      *
73      * This method sends a text package over the network. When the package
74      * terminates to the recipients, they are alerted by the
75      * notifyAboutReceivedTextPackage() method specified by the
76      * FrameworkListener interface.
77      *
78      * @param recipients A list containing the addresses of the recipient nodes
79      * @param textMessage The text message to be sent
80      *
81      */
82     public void sendTextPackage(String[] recipients, String textMessage);
83
84
85     /**
86      *
87      * This method sends a file package over the network. When the package
88      * terminates to the recipients, they are alerted by the
89      * notifyAboutReceivedFilePackage() method specified by the
90      * FrameworkListener interface.
91      *
92      * @param recipients A list containing the addresses of the recipient nodes
93      * @param filePath The path of the file to be sent
94      *
95      */
96     public void sendFilePackage(String[] recipients, String filePath);
97
98
99     /**
100     *
101     * This method returns a list of the files in the given root directory on the device
102     *
103     * @param root The path to the root directory
104     * @return An enumeration containing the names of the files in the root directory
105     */
106    public Enumeration getFileList(String root);
107
108
109
110 }
```

D.1.2 Interface FrameworkListener

```
package peer2me.framework;

import java.util.Hashtable;


/**
 * This interface must be implemented by all Peer2Me MIDlets.
 * It ensures that the Framework can access a set of methods in the MIDlet in order
 * to notify the MIDlet about various events.
 *
 * @author Torbjørn Vatn & Steinar A. Hestnes
 */
public interface FrameworkListener {

  /**
   *
   * This method is called by the framework whenever an exception notice is
     * given by the log. This will be done in cases where exceptions occure
     * in threads and cannot be thrown in the usual way.
   *
   * @param location The location where the Exception occured
   * @param exception The actual Exception
   */
  public void notifyAboutException(String location, Exception exception);


  /**
   *
   * This method is called by the framework when a node is found.
   * These nodes are not yet connected in a network.
   * To do this, use the Framework.connectToNodes() method.
   *
     * @param nodeAddress The network address of the node
     * @param remoteNodeName The name of the found remote node
     */
    public void notifyAboutFoundNode(String nodeAddress, String remoteNodeName);


    /**
     *
     * This method is called from the framework whenever a text package is
     * received from a remote node.
     *
```

```
44      * @param senderName The name of the sender
45      * @param textMessage The received text message
46      */
47     public void notifyAboutReceivedTextPackage(String senderName, String textMessage);
48
49
50     /**
51      *
52      * This method is called from the framework whenever a file package is
53      * received from a remote node.
54      *
55      * @param senderName The name of the sender
56      * @param filePath The path to the received file
57      */
58     public void notifyAboutReceivedFilePackage(String senderName, String filePath);
59
60
61     /**
62      *
63      * This method is called from from the framework to notify the midlet about
64      * the participants of the ad hoc network.
65      *
66      * @param participants A hashtable that contains the names of the participants as unique keys and
67      * the network addresses as values.
68      */
69     public void notifyAboutParticipants(Hashtable participants);
70
71
72
73 }
```

D.1.3 Class FrameworkFrontEnd

```
package peer2me.framework;

import peer2me.network.Network;
import peer2me.util.FileHandler;
import peer2me.util.Log;
import peer2me.domain.FilePackage;
import peer2me.domain.Group;
import peer2me.domain.GroupSyncPackage;
import peer2me.domain.Node;
import peer2me.domain.TextPackage;

import java.io.IOException;
import java.util.Enumeration;
import java.util.Hashtable;


/**
 *
 * This is the main class of the Peer2Me framework. It manages
 * and connects the resources and functions of the framework.
 * It also handles all communication and interaction with the
 * MIDlets running the framework.
 *
 * @author Torbjørn Vatn & Steinar A. Hestnes
 */
public class FrameworkFrontEnd implements Framework{

    // The instance of the FrameworkFrontEnd returned by the getInstance() method, will be casted to Framework upon return
    private static FrameworkFrontEnd singleton;
    // The midlet that initiated the framework represented by a FrameworkListener instance
    private FrameworkListener midlet;
    // The Network instance of the preferred network
    private Network currentNetwork;
    // The group containing all connected nodes running the same application
    private Group group;
    // The local node
    private Node localNode;
    // A Hastable containing the addresses(key) and names(value) of the nodes found in the discovery process
    private Hashtable foundNodes;

    // A Log instance
    private Log log = Log.getInstance();

```

```
44
45  /**
46   *
47   * This method creates an instance of FrameworkFrontEnd and returns it as
48   * a reference of type Framework. This is the only method that can
49   * be called directly from the MIDlet on the FrameworkFrontEnd.
50   * The MIDlet is restricted to only use the methods specified in the
51   * Framework interface.
52   *
53   * @param midlet A reference to the MIDlet (The MIDlet must implement the FrameworkListener interface).
54   * @return A reference to the Framework
55   */
56  public static synchronized Framework getInstance(FrameworkListener midlet){
57    if(singleton == null){
58      singleton = new FrameworkFrontEnd();
59
60      // Creates a instance of the Log class and set self as framework
61      Log.getInstance().setFramework(singleton);
62
63      // Sets the midlet variable
64      singleton.midlet = midlet;
65    }
66
67    // The FrameworkFrontEnd instance are casted to Framework to avoid access to unwanted methods
68    Framework framework = singleton;
69    return framework;
70  }
71
72
73  /**
74   * Constructor. Made private to ensure singleton pattern.
75   */
76  private FrameworkFrontEnd(){}
77
78
79
80  /**
81   *
82   * This method initiates the framework, and is the first method that should
83   * be run after getting a instance of the framework. It initiates the
84   * fundamental services offered by the framework.
85   *
86   * @param nodeName The name of the user of the MIDlet.
87   * @param midletName The name of the MIDlet, eventually translated into a ServiceID used to find other devices
88     * running the same MIDlet.
```

```
 * @param preferredNetwork Deciding which network implementation to use.
 *
 * @throws ClassNotFoundException The input preferredNetwork is invalid
 * @throws IllegalAccessException The input preferredNetwork is invalid
 * @throws InstantiationException The input preferredNetwork is invalid
 * @throws IOException Error initiating framework
 * @throws Exception Error initiating framework
 */
public void initFramework(String nodeName, String midletName, String preferredNetwork) throws ClassNotFoundException,
                                                IllegalAccessException, InstantiationException, IOException, Exception{

  // Creates a Network instance
  currentNetwork = Network.getInstance(preferredNetwork);
    // Sets a reference to this class to be used in the Network class
    currentNetwork.setFrameworkFrontEnd(this);
    // Sets the applicationId to be used by the Network class
    currentNetwork.setApplicationId(midletName);
    // Creates a group that will be filled with nodes running the same application
    group = new Group();
      // Adds a representation of this (the local) node to the group.
      localNode = new Node(nodeName,currentNetwork.getNodeAddress("localnode"));
      group.addParticipant(localNode);
      // Initiates the currentNetwork
      currentNetwork.init();
      // Creates the foundNodes Hashtable
      foundNodes = new Hashtable();
}


/**
 *
 * This method shuts down the framework and closes all the open network connections and streams.
 * It should be called from the MIDlet before closing, to clean up the network connections.
 *
 */
public void shutdownFramework(){
  // Shuts down and closes the Group
  group.shutdownGroup();
  // Shuts down the ConnectionListener
  currentNetwork.getConnectionListener().shutdown();
}


  /**
   *
```

```
134     * This method returns the local representation of the group. It is called from
135     * ConnectionListener.run() or Network.nodeFound() when a remote node is found
136     * and should be added to the group.
137     *
138     * @return The local representation of the group
139     */
140    public Group getGroup(){
141        return group;
142    }
143
144   /**
145    *
146    * This method starts a search for devices running the same MIDlet.
147    * When such a device is found, the notifyAboutFoundNode() method
148    * in this class is called.
149    *
150    * @throws IOException Thrown if the search crashes
151    */
152   public void startNodeSearch() throws IOException{
153    currentNetwork.searchForNodes();
154   }
155
156
157     /**
158     *
159     * This method establishes a connection to the chosen nodes.
160     * After updating the local group, it synchronizes the groups on
161     * all other participating nodes.
162     * The method should be called from the MIDlet.
163     *
164     * @param addresses The addresses to the nodes to connect to.
165     */
166    public void connectToNodes(String[] addresses){
167      // Creates Node objects based on the Vectors nodeNames and nodeAddresses
168      for(int i=0; i<addresses.length; i++){
169        getGroup().addParticipant(new Node((String)foundNodes.get(addresses[i]),addresses[i]));
170      }
171      // Synchronizes the groups on all connected nodes
172      synchronizeGroups();
173    }
174
175
176    /**
177     *
178     * This method is used to make the Framework syncronize the Groups on all the
```

```
179     * connected nodes. The result of running this method is that the method
180     * notifyAboutParticipants() is called on the MIDlet.
181     * It is called from the methods connectToNodes() and
182     * notifyAboutLostNode() in this class.
183     *
184     */
185    private synchronized void synchronizeGroups(){
186
187        // Creates a string table with the recipient addresses
188        Hashtable participatingNodes = group.getParticipatingNodes();
189
190        String [] recipients = new String[0];
191        // Only do this if there is more than this node in the group
192        if (participatingNodes.size()>1){
193        recipients = new String[participatingNodes.size()-1];
194          // Need a list of nodes to run a groupsync
195          Node[] nodes = new Node[participatingNodes.size()];
196          // Adds the local Node to the nodes[]
197          nodes[0] = localNode;
198          // Removes the local Node from the participatingNodes[]
199          participatingNodes.remove(localNode.getAddress());
200
201          Enumeration addresses = participatingNodes.keys();
202          int counter = 0;
203
204          while(addresses.hasMoreElements()){
205                String address = (String)addresses.nextElement();
206                // Does not add the local node
207                recipients [counter] = address;
208                // Fetches the Node objects from participatingNodes
209                nodes[counter+1] = (Node)participatingNodes.get(address);
210                counter++;
211          }
212
213          // Sends a networkpackage to all participants to synchronize the group on all nodes
214          if ( recipients .length!=0){
215                currentNetwork.sendDataPackage(new GroupSyncPackage(localNode,recipients,nodes),recipients);
216            }
217          // Adds the local Node to the group again
218          participatingNodes.put(localNode.getAddress(),localNode);
219      }
220
221        // Notifies the MIDlet about the participants of the group
222        notifyAboutParticipants();
223
```

```
224         // Logs the sending of the data package
225         String recipientNames = "";
226         for(int i=0; i<recipients.length; i++){
227           if(group.getNode(recipients[i])!=null){
228               recipientNames +="- "+group.getNode(recipients[i]).getNodeName()+" ("+recipients[i]+") \n";
229             }
230         }
231 
232         if(recipients.length>0)log.logDataPackage("Sent a group sync package to:\n "+recipientNames);
233     }
234 
235 
236     /**
237      *
238      * This method returns a reference to the local node.
239      *
240      * @return An object representing the local node
241      */
242     public Node getLocalNode(){
243         return localNode;
244     }
245 
246      /**
247      *
248      * This method is used by the MIDlet to send a text package over the network.
249      * When the package terminates to the recipients, the
250      * notifyAboutReceivedTextPackage() method in this class is run.
251      *
252      * @param recipients A list containing the addresses of the recipient nodes
253      * @param textMessage The text to be sent
254      *
255      */
256     public void sendTextPackage(String[] recipients, String textMessage){
257 
258       // Logs the sending of the text package
259         String recipientNames = "";
260         for(int i=0; i<recipients.length; i++){
261           recipientNames +="- "+group.getNode(recipients[i]).getNodeName()+" ("+recipients[i]+") \n";
262         }
263         log.logDataPackage("Sending textpackage to:\n"+recipientNames);
264 
265         TextPackage textPackage = new TextPackage(localNode,recipients,textMessage);
266         // Passes the task of sending the data package over to the network
267         if(recipients.length!=0)currentNetwork.sendDataPackage(textPackage, recipients);
268     }
```

```
269
270
271     /**
272      *
273      * This method is used by the MIDlet to send a file package over the network.
274      * When the package terminates to the recipients, the
275      * notifyAboutReceivedFilePackage() method in this class is run.
276      *
277      * @param recipients A list containing the addresses of the recipient nodes
278      * @param filePath The path of the file to send
279      *
280      */
281     public void sendFilePackage(String[] recipients, String filePath){
282         FilePackage filePackage = new FilePackage(localNode,recipients,filePath);
283     // Passes the task of sending the data package over to the network
284     if(recipients.length!=0)currentNetwork.sendDataPackage(filePackage, recipients);
285
286         // Logs the sending of the file package
287         String recipientNames = "";
288         for(int i=0; i<recipients.length; i++){
289           recipientNames += group.getNode(recipients[i]).getNodeName()+" ("+recipients[i]+") \n";
290         }
291         log.logDataPackage("Sending file to: "+recipientNames);
292     }
293
294
295     /**
296      *
297      * This method returns a list of the files in the given root directory on the device
298      *
299      * @param root The path to the root directory
300      * @return A Enumeration containing the names of the files in the root directory
301      */
302     public Enumeration getFileList(String root){
303       // Creates a FileHandler representing the root
304       FileHandler fileHandler = new FileHandler(root);
305       // Returns the file list of the root
306       Enumeration list = fileHandler.getFileList();
307       return list;
308     }
309
310
311   /*****************************************************************/
312   // The notify methods used to notify the midlet about various events
313     /*****************************************************************/
```

```
314
315     /**
316      *
317      * This method is called from the nodeFound() method in the Network class whenever a node is found
318      *
319      * @param address The network address of the node
320        * @param remoteNodeName The name of the found remote node
321      */
322     public void notifyAboutFoundNode(String address, String remoteNodeName){
323
324       // Here we add a number after equal node names to make them unique
325         // We do this so we can set the node names as keys and the node addresses as values
326         // The reason for doing this is that the node names will be displayed in the midlet
327         // and after selecting a node name, the address should be sent to the framework.
328         if (foundNodes.contains(remoteNodeName) || remoteNodeName.equals(localNode.getNodeName())){
329             for (int i=-1;i<foundNodes.size();i++){
330               if (!foundNodes.contains(remoteNodeName+" "+(i+2))){
331                 remoteNodeName = remoteNodeName+" "+(i+2);
332                 i = foundNodes.size();
333               }
334             }
335         }
336
337       // Stores the address and the name of the node in the foundnodes table
338       foundNodes.put(address, remoteNodeName);
339       midlet.notifyAboutFoundNode(address,remoteNodeName);
340     }
341
342
343     /**
344      * This method removes a lost node from the group.
345      * It is called from Network.sendDataPackage() if a node is unreachable.
346      * After removing the node, the groups on all other nodes become
347      * synchronized.
348      *
349      * @param address The address to the lost node
350      */
351     public synchronized void notifyAboutLostNode(String address){
352         log.logDataPackage(getGroup().getNode(address).getNodeName()+" ("+address+") not reachable");
353       log.logConnection("Disconnected "+getGroup().getNode(address).getNodeName()+" ("+address+")");
354       getGroup().removeParticipant(address);
355       // Synchronizes the groups on the connected devices
356       synchronizeGroups();
357     }
358
```

```
359     /**
360      *
361      * This method passes on the Exception notice from the Log to the MIDlet.
362        * This will be done in cases where exceptions occure in threads and
363        * cannot be thrown in the usual way.
364      *
365      * @param location The location (class and method) where the Exception occured
366      * @param exception The actual Exception
367      */
368     public void notifyAboutException(String location, Exception exception){
369       midlet.notifyAboutException(location, exception);
370     }
371
372       /**
373        *
374        * This method is called from NodeConnection.processIncomingData()
375        * whenever a groupSyncPackage is received from a remote node.
376        * The method processes the package, logs the event, and updates the group.
377        *
378        * @param groupSyncPackage The received groupSyncPackage.
379        */
380       public void notifyAboutReceivedGroupSyncPackage(GroupSyncPackage groupSyncPackage){
381
382           // Resets the group before synch
383           group.removeAllParticipants();
384
385           // Uses the content of the package to update the group on this device
386           Node[] participants = groupSyncPackage.getParticipants();
387
388           for(int i=0; i<participants.length; i++){
389               group.addParticipant(participants[i]);
390           }
391           notifyAboutParticipants();
392
393           String sender = groupSyncPackage.getSender().getAddress();
394           log.logDataPackage("Received group sync package from "+sender);
395       }
396
397
398       /**
399        *
400        * This method is called from NodeConnection.processIncomingData()
401        * whenever a text package is received from a remote node.
402        * It processes the package, logs the event, and notifies the midlet.
403        *
```

```
404      * @param textPackage The received text package.
405      */
406     public void notifyAboutReceivedTextPackage(TextPackage textPackage){
407         log.logDataPackage("Received text package from "+textPackage.getSender().getNodeName()+".");
408         midlet.notifyAboutReceivedTextPackage(textPackage.getSender().getNodeName(), textPackage.getContent());
409     }
410
411     /**
412      *
413      * This method is called from NodeConnection.processIncomingData()
414      * whenever a file package is received from a remote node.
415      * It processes the package, logs the event, and notifies the midlet.
416      *
417      * @param filePackage The received file package.
418      */
419     public void notifyAboutReceivedFilePackage(FilePackage filePackage){
420       log.logDataPackage("Received file ("+filePackage.getFilePath()+") from "+filePackage.getSender().getNodeName());
421       midlet.notifyAboutReceivedFilePackage(filePackage.getSender().getNodeName(), filePackage.getFilePath());
422     }
423
424     /**
425      *
426      * This method notifies the midlet about the current group by running the
427      * notifyAboutParticipants method. It is e.g. called from
428      * FrameworkFrontEnd.synchronizeGroups().
429      *
430      */
431     private void notifyAboutParticipants(){
432         // Fetches all the participating Nodes
433         midlet.notifyAboutParticipants(getGroup().getParticipatingNodeNames(this));
434     }
435
436 }
```

D.2 Package peer2me.domain

D.2.1 Class DataPackage

```
package peer2me.domain;

import peer2me.util.Log;


/**
 *
 * This class is the super class of the different type of packages that can be
 * sent between nodes in the network. It contains the attributes that are common
 * for all types of data packages. These are the address of the sender and the
 * address(es) to the recipiant(s) of the DataPackage. Currently, there exists
 * three types of data packages.
 *
 * @author Torbjørn Vatn & Steinar A. Hestnes
 */
public abstract class DataPackage {

  // A Log instance
  public Log log = Log.getInstance();

    // The type of package
    private int type;
  // The Node that sendt the data package
  private Node sender;
  // The addresses of the nodes that are the recipients of the data package
  private String[] recipients;

  /* The constants representing the different types of data packages used
    in node connection to determine the type of package recived */
    public final static int GROUP_SYNC_PACKAGE = 0;
    public final static int TEXT_PACKAGE = 1;
    public final static int FILE_PACKAGE = 2;

  /**
   *
   * Constructor
   *
   * @param type The type specifying the type of data package
   * @param sender A node object representing the sender node
   * @param recipients The addresses to the recipients of the data package
   *
```

```
42      */
43     public DataPackage(int type, Node sender, String[] recipients){
44             this.type = type;
45             this.sender = sender;
46       this.recipients = recipients;
47     }
48
49     /**
50      *
51      * Constructor used to create an empty DataPackage object to fill with the
52      * parseBytes() method
53      *
54      * @param type The type of the DataPackage
55      */
56     public DataPackage(int type){
57       this.type = type;
58     }
59
60
61       /**
62        *
63        * This method returns an int indicating the type of data package
64        *
65        * @return type An int indicating the type of data package
66        */
67       public int getType(){
68           return type;
69       }
70
71
72     /**
73      *
74      * This method returns the sender of this data package
75      *
76      * @return sender The node that sends this package
77      */
78     public Node getSender(){
79       return sender;
80     }
81
82
83       /**
84        *
85        * This method sets the sender of this data package
86        *
```

```
87      * @param sender The node that sends this package
88      */
89     public void setSender(Node sender){
90         this.sender = sender;
91     }
92
93
94
95     /**
96      *
97      * This method returns all the recipients of this data package
98      *
99      * @return recipients The addresses to the recipients of this package
100     */
101    public String[] getRecipients(){
102        return recipients;
103    }
104
105
106     /**
107      *
108      * This method sets the nodes to receive this package
109      *
110      * @param recipients The addresses to the nodes that shall receive this package
111      */
112     public void setRecipients(String[] recipients){
113         this.recipients = recipients;
114     }
115
116
117    /*****************************************************************************/
118    // The abstract methods inherited and overridden by the sub data package classes
119    /*****************************************************************************/
120
121    /**
122     *
123     * This method transforms this data package into a byte array (byte[]) that
124     * is possible to send over a network stream
125     *
126     * @return The byte[] representation of the data package
127     */
128    public abstract byte[] toSendableFormat();
129
130    /**
131     *
```

```
132     * This method parses the content of the byte array (byte[]) back into a DataPackage object
133     *
134     * @param data The byte[] containing the data representing the DataPackage object
135     *
136     */
137    public abstract void parseBytes(byte[] data);
138
139 }
```

D.2.2 Class TextPackage

```
package peer2me.domain;

import java.util.Vector;

/**
 *
 * This class represents a data package containing text that should be
 * sent over the network.
 *
 * @author Torbjørn Vatn & Steinar A. Hestnes
 */
public class TextPackage extends DataPackage {

  // The String content of this TextPackage
  private String content;


  /**
   *
   * Constructor
   *
   * @param sender A node object representing the sender node
   * @param recipients The addresses to the recipients of the text package
   * @param content The String to be sent
   */
  public TextPackage(Node sender, String[] recipients, String content){
    super(TEXT_PACKAGE, sender, recipients);
    this.content = content;
  }

  /**
   *
   * Constructor used to create an empty TextPackage object to fill with the
   * parseBytes() method
   */
  public TextPackage(){
    super(TEXT_PACKAGE);
  }

  /**
   *
   * This method returns the text content of this TextPackage
   *
```

```
44      * @return The content
45      */
46     public String getContent(){
47       return content;
48     }
49
50     /**
51      *
52      * This method transforms this text package into a byte array (byte[]) that
53      * is possible to send over a network stream
54      *
55      * @return The byte[] representation of the text package
56      *
57      */
58     public byte[] toSendableFormat() {
59
60       // The String to send
61           String sendableFormat = "";
62
63       // Setting the sender in the sendableFormat String
64       // The format of the String is:
65       // -> from:"address-of-the-sender":"name-of-the-sender"
66       //
67       sendableFormat += "from:"+getSender().getNodeName()+":"+getSender().getAddress()+"\n";
68
69       // Setting the recipients in the sendableFormat String
70       // The format of the String is:
71       // -> to:"address-of-the-recipient"
72       //
73       String[] recipients = getRecipients();
74       for(int i=0; i<recipients.length; i++){
75           sendableFormat += "to:"+recipients[i]+"\n";
76       }
77
78       // Setting the content in the sendableFormat String
79       // The format of the String is:
80       // -> content:"content"
81       //
82       sendableFormat += "content:"+content+"\n";
83
84       return sendableFormat.getBytes();
85     }
86
87
88     /**
```

```
89      *
90      * This method parses the content of the byte array (byte[]) back into a TextPackage object
91      *
92      * @param data The byte[] containing the data representing the TextPackage object
93      */
94     public void parseBytes(byte[] data){
95       // Counter that keeps track of how many bytes are converted
96       int processed = 0;
97       // The node addresses found in the "to" section of the String
98       Vector recipients = new Vector();
99           char newLine = '\n';
100      // The loop processing the bytes
101      while(processed < data.length){
102        // The StringBuffer temporary holding the content of the byte[]
103        StringBuffer buffer = new StringBuffer();
104        // Fetches the content of the byte[], stops for every new line (marked with a "\n")
105        while(data[processed] != newLine){
106          buffer.append((char)data[processed]);
107          processed++;
108        }
109
110        // Retrives the sender, marked by "from"
111        if(buffer.toString().startsWith("from")){
112          int fromStart = buffer.toString().indexOf(":")+1;
113          try{
114            Node from = Node.restoreNode(buffer.toString().substring(fromStart));
115            setSender(from);
116          }catch(Exception e){
117            log.logException("TextPackage.parseBytes()",e,false);
118          }
119        }else
120
121        if(buffer.toString().startsWith("to")){
122          int toStart = buffer.toString().indexOf(":")+1;
123          try{
124            recipients.addElement(buffer.toString().substring(toStart));
125          }catch(Exception e){
126            log.logException("TextPackage.parseBytes()",e,false);
127          }
128        }else
129        if(buffer.toString().startsWith("content")){
130          int contentStart = buffer.toString().indexOf(":")+1;
131          content = buffer.toString().substring(contentStart);
132        }
133        // Adds 1 to the counter that keeps track of the bytes processed,
```

```
134                 // this is added due to the \n
135         processed++;
136
137         // Sets the recipients in the super class DataPackage
138         String[] recipientAddreses = new String[recipients.size()];
139                 recipients.copyInto(recipientAddreses);
140         setRecipients(recipientAddreses);
141
142     }
143   }
144
145 }
```

D.2.3 Class FilePackage

```
package peer2me.domain;

import java.util.Vector;
import peer2me.util.FileHandler;

/**
 * This class represents a data package containing metadata about a file of some
 * sort that should be sent over the network. The package contains the file path
 * and length of the file to transfer, so that the receiver can handle the incoming
 * stream of data and transform it back into a copy of the file.
 *
 * @author Torbjørn Vatn & Steinar A. Hestnes
 */
public class FilePackage extends DataPackage {

  // The file path of the file to transfer
  private String filePath;

  // The size of the file
  private long fileSize;


  /**
   *
   * Constructor
   *
   * @param sender A node object representing the sender node
   * @param recipients The addresses to the recipients of the file package
   * @param filePath The path of the file to be sent
   */
  public FilePackage(Node sender, String[] recipients, String filePath){
    super(FILE_PACKAGE,sender,recipients);
    this.filePath = filePath;
    this.fileSize = new FileHandler(filePath).getFileSize();
  }

  /**
   *
   * Constructor used to create an empty FilePackage object to fill with the
   * parseBytes() method
   *
   */
  public FilePackage(){
```

```
44        super(FILE_PACKAGE);
45    }
46
47    /**
48     *
49     * This method returns the file path of this FilePackage
50     *
51     * @return The file path
52     */
53    public String getFilePath(){
54        return filePath;
55    }
56
57    /**
58     *
59     * This method returns the file size of this FilePackage
60     *
61     * @return The file size
62     */
63    public long getFileSize(){
64        return fileSize;
65    }
66
67
68
69    /**
70     *
71     * This method transforms this file package into a byte array (byte[]) that
72     * is possible to send over a network stream
73     *
74     * @return The byte[] representation of the file package
75     *
76     */
77    public byte[] toSendableFormat() {
78
79        // The String to send
80        String sendableFormat = "";
81
82        // Seting the sender in the sendableFormat String
83        // The format of the String is:
84        // -> from:"address-of-the-sender":"name-of-the-sender"
85        //
86        sendableFormat += "from:"+getSender().getNodeName()+":"+getSender().getAddress()+"\n";
87
88        // Setting the recipients in the sendableFormat String
```

```
    // The format of the String is:
    // -> to:"address-of-the-recipient"
    //
    String[] recipients = getRecipients();
    for(int i=0; i<recipients.length; i++){
        sendableFormat += "to:"+recipients[i]+"\n";
    }

    // Setting the filePath in the sendableFormat String
    // The format of the String is:
    // -> filePath:"filePath"
    //
    sendableFormat += "filePath:"+filePath+"\n";

    // Setting the fileSize in the sendableFormat String
    // The format of the String is:
    // -> fileSize:" fileSize "
    //
    sendableFormat += "fileSize:"+fileSize+"\n";

    return sendableFormat.getBytes();
  }

  /**
   *
   * This method parses the content of the byte array (byte[]) back into a FilePackage object
   *
   * @param data The byte[] containing the data representing the FilePackage object
   */
  public void parseBytes(byte[] data){
    // Counter that keeps track of how many bytes are converted
    int processed = 0;
    // The node addresses found in the "to" section of the String
    Vector recipients = new Vector();
        char newLine = '\n';
    // The loop processing the bytes
    while(processed < data.length){
      // The StringBuffer temporary holding the content of the byte[]
      StringBuffer buffer = new StringBuffer();
      // Fetches the content of the byte[], stops for every new line (marked with a "\n")
      while(data[processed] != newLine){
        buffer.append((char)data[processed]);
        processed++;
      }

```

```
134         // Retrives the sender, marked by "from"
135         if(buffer.toString().startsWith("from")){
136           int fromStart = buffer.toString().indexOf(":")+1;
137           try{
138             Node from = Node.restoreNode(buffer.toString().substring(fromStart));
139             setSender(from);
140           }catch(Exception e){
141             log.logException("FilePackage.parseBytes()",e, false );
142           }
143         }else
144
145         if(buffer.toString().startsWith("to")){
146           int toStart = buffer.toString().indexOf(":")+1;
147           try{
148             recipients.addElement(buffer.toString().substring(toStart));
149           }catch(Exception e){
150             log.logException("FilePackage.parseBytes()",e, false );
151           }
152         }else
153
154         if(buffer.toString().startsWith("filePath")){
155           int contentStart = buffer.toString().indexOf(":")+1;
156           filePath = buffer.toString().substring(contentStart);
157         }else
158
159         if(buffer.toString().startsWith(" fileSize ")){
160           int contentStart = buffer.toString().indexOf(":")+1;
161           fileSize = Long.parseLong(buffer.toString().substring(contentStart));
162         }
163         // Adds 1 to the counter that keeps track of the bytes processed,
164               // this is added due to the \n
165         processed++;
166
167         // Sets the recipients in the super class DataPackage
168         String[] recipientAddreses = new String[recipients.size()];
169               recipients.copyInto(recipientAddreses);
170         setRecipients(recipientAddreses);
171
172       }
173     }
174 }
```

D.2.4 Class GroupSyncPackage

```
package peer2me.domain;

import java.util.Vector;

/**
 *
 * A GroupSyncPackage is a package used internally in the
 * framework to synchronize the groups containing the participants. The participant performing
 * the groupsync uses its own group as content of the package. All the receivers synchronizes
 * their groups based on the information found in the GroupSyncPackage.
 *
 * @author Torbjørn Vatn & Steinar A. Hestnes
 */
public class GroupSyncPackage extends DataPackage {

    // The participating nodes
    private Node[] participatingNodes;


  /**
   *
   * Constructor
   *
   * @param sender A node object representing the sender node
   * @param recipients The addresses to the recipients of the groupsync package
   * @param participatingNodes A hashtable with node addresses as keys and names as values
   *
   */
  public GroupSyncPackage(Node sender, String[] recipients, Node[] participatingNodes){
    super(GROUP_SYNC_PACKAGE, sender, recipients);
    this.participatingNodes = participatingNodes;
  }


  /**
   *
   * Constructor used to create an empty GroupSyncPackage object to fill
   * with the parseBytes() method
   */
  public GroupSyncPackage(){
    super(GROUP_SYNC_PACKAGE);
  }

```

```
44
45      /**
46       *
47       * This method returns a list of the nodes that are participating in the network (group)
48       *
49       * @return A list of participating nodes
50       */
51      public Node[] getParticipants(){
52          return participatingNodes;
53      }
54
55    /**
56     *
57     * This method transforms this groupsync package into a byte array (byte[])
58     * that is possible to send over a network stream
59     *
60     * @return The byte[] representation of the groupsync package
61     *
62     */
63    public byte[] toSendableFormat() {
64
65      // The String to send
66      String sendableFormat = "";
67
68      // Setting the sender in the sendableFormat String
69      // The format of the String is:
70      // -> from:"address-of-the-sender":"name-of-the-sender"
71      //
72      Node sender = getSender();
73      sendableFormat += "from:"+sender.getNodeName()+":"+sender.getAddress()+"\n";
74
75      // Setting the recipients in the sendableFormat String
76      // The format of the String is:
77      // -> to:"address-of-the-recipient"
78      //
79      String[] recipients = getRecipients();
80      for(int i=0; i<recipients.length; i++){
81          sendableFormat += "to:"+recipients[i]+"\n";
82      }
83
84      // Setting the content in the sendableFormat String
85      // The format of the String is:
86      // -> participant:"name-of-a-participant":"address-of-a-participant"
87      //
88      for(int i=0; i<participatingNodes.length; i++){
```

```
89          sendableFormat += "participant:"+participatingNodes[i].getNodeName()+":"+participatingNodes[i].getAddress()+"\n";
90          }
91
92      return sendableFormat.getBytes();
93  }
94
95  /**
96   *
97   * This method parses the content of the byte array (byte[]) back into a
98   * GroupSyncPackage object
99   *
100  * @param data The byte[] containing the data representing the GroupSyncPackage object
101  */
102 public void parseBytes(byte[] data) {
103     // Counter that keeps track of how many bytes are converted
104     int processed = 0;
105     // The node addresses used in the "to" section of the package
106         Vector recipients = new Vector();
107         // The node addresses used in the "participant" section of the package
108         Vector participants = new Vector();
109
110         char newLine = '\n';
111     // The loop processing the bytes
112     while(processed < data.length){
113         // The StringBuffer temporary holding the content of the byte[]
114         StringBuffer buffer = new StringBuffer();
115         // Fetches the content of the byte[], stops for every new line (marked with a "\n")
116         while(data[processed] != newLine){
117             buffer.append((char)data[processed]);
118             processed++;
119         }
120
121                 // Adds 1 to the counter that keeps track of the bytes processed,
122                 // this is added due to the \n (new line)
123                 processed++;
124
125                 // Runs the toString() on the buffer
126                 String line = buffer.toString();
127
128         // Retrives the sender, marked by "from"
129                 if(line.startsWith("from")){
130             int fromStart = line.indexOf(":")+1;
131             try{
132                 Node from = Node.restoreNode(line.substring(fromStart));
133                 setSender(from);
```

```
134         }catch(Exception e){
135           log.logException("NetworkPackage.parseBytes()1",e,false);
136         }
137       }else
138
139                 // Retrives the recipients, marked by "to"
140                 if(line.startsWith("to")){
141           int toStart = line.indexOf(":")+1;
142           try{
143             recipients.addElement(line.substring(toStart));
144           }catch(Exception e){
145             log.logException("NetworkPackage.parseBytes()2",e,false);
146           }
147           // Sets the recipients in the super class DataPackage
148           String[] recipientAddresses = new String[recipients.size()];
149                 recipients.copyInto(recipientAddresses);
150           setRecipients(recipientAddresses);
151       }else
152
153           // Retrives the participants, marked by "participant" on each line
154           if(line.startsWith("participant")){
155                 int participantStart = line.indexOf(":")+1;
156                 try{
157                     Node participant = Node.restoreNode(line.substring(participantStart));
158                     participants.addElement(participant);
159                 }catch(Exception e){
160                     log.logException("NetworkPackage.parseBytes()3",e,false);
161                 }
162           }
163     }
164         // End of package (while-loop). Adds the found participants to a list that can be read from the package object
165         participatingNodes = new Node[participants.size()];
166         participants.copyInto(participatingNodes);
167   }
168 }
```

D.2.5 Class Group

```
package peer2me.domain;

import java.util.Hashtable;
import java.util.Enumeration;
import peer2me.framework.FrameworkFrontEnd;
import peer2me.network.NodeConnection;


/**
 * This class represents a group of nodes running the same service (MIDlet).
 * All connected nodes in the ad hoc network are participants in the group.
 * Participants can be added and removed, and a list of all the
 * participants can be retreived.
 *
 * @author Torbjørn Vatn & Steinar A. Hestnes
 */

public class Group {

    // A list containing participating nodes
    private Hashtable participatingNodes;


    /**
     *
     * Constructor.
     *
     * Creates a new Group.
     * A group is created in FrameworkFrontEnd.initFramework().
     */
    public Group(){
        participatingNodes = new Hashtable();
    }

    /**
     *
     * This method closes the NodeConnection of all the participating nodes, and
     * removes all nodes from the group.
     * It is called from the MIDlet via FrameworkFrontEnd.shutdownFramework()
     * when all network connections should be closed.
     *
     */
    public void shutdownGroup(){
```

```
44          Enumeration nodes = participatingNodes.elements();
45          while(nodes.hasMoreElements()){
46            Node node = (Node)nodes.nextElement();
47            // Closes the connection
48            NodeConnection connection = node.getNodeConnection();
49            if (connection != null)connection.closeConnection();
50            // Removes the node from the Group
51              participatingNodes.remove(node.getAddress());
52          }
53        }
54
55
56        /**
57         *
58         * This method adds a node to the group as a participant.
59         *
60         * @param node The node to add as a participant.
61         */
62        public void addParticipant(Node node){
63            // Adds the node only if it is not added already.
64
65            // This test is necessary during groupsync
66            if (!participatingNodes.containsKey(node.getAddress())){
67              participatingNodes.put(node.getAddress(),node);
68            }else{
69                // If the node already exists in the participant list , this is the node that
70                // initially discovered this node and was saved only with address and connection
71                // Name is still missing and we have to add it
72                if (node.getNodeName() != null){
73                  ((Node)participatingNodes.get(node.getAddress())).setNodeName(node.getNodeName());
74                }
75
76                if (node.getNodeConnection() != null){
77                    if (node.getNodeConnection().getConnection() != null){
78                        // Important to start the connection!
79                        ((Node)participatingNodes.get(node.getAddress())).startNodeConnection();
80                      ((Node)participatingNodes.get(node.getAddress())).getNodeConnection().setConnection(
81                                                          node.getNodeConnection().getConnection());
82                    }
83                }
84            }
85        }
86
87
88        /**
```

```
89      *
90      * This method removes a participating node.
91      *
92      * @param address The address of the node to remove from this group
93      */
94     public void removeParticipant(String address){
95         participatingNodes.remove(address);
96     }
97
98
99     /**
100     * This method removes all participating nodes.
101     * It is used to clear the group before it is updated by a
102     * groupSyncPackage received from a remote node.
103     *
104     */
105    public void removeAllParticipants(){
106      participatingNodes.clear ();
107    }
108
109
110    /**
111     *
112     * This method returns a list containing the nodes participating in this group.
113     * The address is the key to find the Node.
114     *
115     * @return A list containing the nodes participating in this group. The address is
116     * the key and the node name is the value
117     */
118    public Hashtable getParticipatingNodes(){
119        return participatingNodes;
120    }
121
122    /**
123     *
124     * This method returns a list containg the names (as keys) of the nodes participating in this group.
125     * The addresses are stored as values.
126     * It is called from FrameworkFrontEnd.notifyAboutParticipants().
127     *
128     * @return A list containing the nodes participating in this group. The node name is
129     * the key and the address is the value
130     */
131    public Hashtable getParticipatingNodeNames(FrameworkFrontEnd frameworkFrontEnd){
132
133      // An enum containing the node addresses
```

```
        Enumeration addresses = participatingNodes.keys();

        // Makes a hashtable with names as keys and addresses as values
        Hashtable names = new Hashtable();

        while(addresses.hasMoreElements()){
            String address = (String)addresses.nextElement();
            // The local node should not be added because a user should not send datapackages to him/her-self
            if (!address.equals(frameworkFrontEnd.getLocalNode().getAddress())){
                Node node = (Node)participatingNodes.get(address);
                String name = node.getNodeName();
                names.put(name,address);
            }
        }
        return names;
    }


    /**
     *
     * This method returns a node with the address specified as input
     *
     * @param address The address of the node to get
     * @return A node with the address specified as input
     */
    public Node getNode(String address){
        Object node = participatingNodes.get(address);
        if (node != null) return (Node)participatingNodes.get(address);
        else return null;
    }


}
```

D.2.6 Class Node

```
package peer2me.domain;

import peer2me.network.NodeConnection;
import javax.microedition.io.StreamConnection;


/**
 * This class represents a node in the network.
 * It contains information like the name of the node and its network address.
 * A node also owns a nodeConnection object listening for- and processing
 * incoming and outgoing data packages.
 *
 * @author Torbjørn Vatn & Steinar A. Hestnes
 */
public class Node {

    // Object variables relevant for a node
    private String nodeName;
    private String address;
    // The connection to the remote node
    private StreamConnection connection;
    // The NodeConnection holding the connection to the remote node
    private NodeConnection nodeConnection;


    /**
     *
     * Constructor. Creates a new Node.
     * This constructor is used when a node is created to represent the LOCAL
     * device. In this case, nodeName and address are known.
     * The constructor is called from FrameworkFrontEnd.initFramework().
     *
     * @param nodeName The name of the node
     * @param address The node network address
     */
    public Node(String nodeName, String address){
        this.nodeName = nodeName;
        this.address = address;
    }


    /**
     *
```

```
44      * Constructor. Creates a new Node.
45      * This constructor is used when a node is created to represent a remote
46      * device on the node which INITIATED the search.
47      * In this case, name and address is known. In addition, a
48      * StreamConnection object containing a connection to this remote device
49      * exists.
50      * The constructor is called from the nodeFound() method in the Network subclass.
51      *
52      * @param nodeName The name of the node
53      * @param address The node network address
54      * @param connection The connection to this remote node
55      *
56      */
57     public Node(String nodeName, String address, StreamConnection connection){
58         this.nodeName = nodeName;
59         this.address = address;
60         this.connection = connection;
61         // Starts the connection thread on the remote node
62         startNodeConnection();
63     }
64
65
66     /**
67      *
68      * Constructor. Creates a new Node.
69      * This constructor is used when a node is created to represent a remote
70      * device on the node which was DISCOVERED during a search.
71      * In this case, only the address is known. In addition, a
72      * StreamConnection object containing a connection to this remote device
73      * exists.
74      * The constructor is called from the run() method in ConnectionListener.
75      *
76      * @param address The node network address
77      * @param connection The connection to this remote node
78      */
79     public Node(String address, StreamConnection connection){
80         this.address = address;
81         this.connection = connection;
82         // Starts the connection thread on the remote node
83         startNodeConnection();
84     }
85
86
87     /**
88      *
```

```
89      * This method creates a nodeConnection running two threads.
90      * One of the threads listens for incoming data packages, and the other
91      * processes outgoing data packages.
92      * It is only used when this node object represents a remote node.
93      *
94      */
95     public void startNodeConnection(){
96       // Starts a thread that listens for incoming and outgoing messages from/to this node
97       if (nodeConnection == null) nodeConnection = new NodeConnection(connection, this);
98     }
99
100    /**
101     *
102     * This method returns the NodeConnection owned by this node
103     *
104     * @return nodeConnection This nodes NodeConnection
105     */
106    public NodeConnection getNodeConnection(){
107        return nodeConnection;
108    }
109
110    /**
111     * This method sets the connection to this remote node.
112     * It is called from Network.nodeFound().
113     *
114     * @param connection The connection to this remote node
115     */
116    public void setNodeConnection(StreamConnection connection){
117        this.connection = connection;
118        if (nodeConnection != null) nodeConnection.setConnection(connection);
119    }
120
121   /**
122    *
123    * This method returns the name of the node
124    *
125    * @return The nodeName
126    */
127    public String getNodeName() {
128        return nodeName;
129    }
130
131
132    /**
133     *
```

```
     * This method sets the name of the node
     *
     * @param nodeName The name of the node
     */
    public void setNodeName(String nodeName){
        this.nodeName = nodeName;
    }


    /**
     *
     * This method returns the node address
     *
     * @return The node network address
     */
    public String getAddress() {
        return address;
    }


    /**
     *
     * This method restores a node with the properties specified in the given input string.
     *
     * @param nodeString A string containing node properties (name:address)
     *
     */
    public static Node restoreNode(String nodeString){
        int separator = nodeString.indexOf(":");
        return new Node(nodeString.substring(0, separator),nodeString.substring(separator+1, nodeString.length()));
    }


}
```

D.3 Package peer2me.util

D.3.1 Class Log

```
package peer2me.util;

import peer2me.framework.FrameworkFrontEnd;
import java. util .Vector;



/**
 * This class contains functionality to create and maintain a log of events
 * and exceptions.
 * The Log contains four differnet kinds of logs, an exception log, a connection
 * log, a data package log and a debug log. They can be used to log events from anywhere in
 * the framework, and the logs can be retreived later to get information about the execution
 * of the MIDlet.
 *
 * @author Torbjørn Vatn & Steinar A. Hestnes
 */
public class Log {

  // The singleton instance of this class
  private static Log singleton;
  // This FrameworkFrontEnd instance enables the Log to notify the Framework of exceptions
  FrameworkFrontEnd framework;


  // These Vectors are used to store the different elements we want to log
  /*******************************************************************/
  // The Vector containing the logged Exceptions
  private Vector exceptionLog = new Vector();
  // The Vector containing the opened Connections
  private Vector connectionLog = new Vector();
    // The Vector containing information about exchanged data packages
    private Vector dataPackageLog = new Vector();
    // The Vector containing debug information
    private Vector debugLog = new Vector();

  // These static variables represent the different Log Vectors
  /*********************************************************/
  public static final int EXCEPTION_LOG = 1;
  public static final int CONNECTION_LOG = 2;
    public static final int DATA_PACKAGE_LOG = 3;
```

```
42      public static final int DEBUG_LOG = 4;
43
44
45    /**
46     *
47     * This method returns the only existing instance of the Log class
48     *
49     * @return The singleton instance of the Log class
50     */
51    public static synchronized Log getInstance(){
52      if (singleton == null){
53        singleton = new Log();
54      }
55      return singleton;
56    }
57
58    /**
59     *
60     * Constructor. Made private to ensure singleton pattern.
61     *
62     */
63    private Log(){};
64
65    /**
66     *
67     * This method is called by the FrameworkFrontEnd to reveal itself to the Log
68     *
69     * @param framework The FrameworkFrontEnd refrence sent by the FrameworkFrontEnd itself
70     */
71    public void setFramework(FrameworkFrontEnd framework){
72      this.framework = framework;
73    }
74
75    /**
76     *
77     * This method adds an Exception entry to the Exception log
78     *
79     * @param location The location (class and method) where the Exception occured
80     * @param exception The actual Exception
81     * @param notify This boolean decides whether or not to notify the Framework about the Exception that occured
82     */
83    public void logException(String location, Exception exception, boolean notify){
84
85      // The actual String stored in the log
86      String logString = exception.getMessage()+" @ "+location;
```

```
87
88      // Adding the exception message to the log
89      exceptionLog.addElement(logString);
90
91      // Notifying the Framework of the Exception if requested
92      if(notify) framework.notifyAboutException(location, exception);
93    }
94
95
96      /**
97       *
98       * This method adds a Connection entry to the Connection log
99       *
100      * @param connectionStatus A textual description of the connection status
101      */
102     public void logConnection(String connectionStatus){
103         connectionLog.addElement(connectionStatus);
104     }
105
106
107     /**
108      *
109      * This method adds a data package entry to the data package log
110      *
111      * @param packageStatus A textual description of the data package status
112      */
113     public void logDataPackage(String packageStatus){
114       dataPackageLog.addElement(packageStatus);
115     }
116
117     /**
118      *
119      * This method adds a Debug entry to the Debug log
120      *
121      * @param location The location (class and method) where the debuginfo was logged
122      * @param debugInfo A textual description of the debug information
123      */
124     public void logDebugInfo(String location, String debugInfo){
125
126         // The actual String stored in the log
127         String logString = debugInfo+" @ "+location;
128
129         debugLog.addElement(logString);
130     }
131
```

```
132     /**
133      *
134      * This method returns the desired log in a displayable format.
135      * It must be called by activating a soft button in the application.
136      * Due to multithreading and interuptions there can be some delay in the creation of the log.
137      *
138      * @param log The Log.FIELD representing the desired log
139      * @return The desired log as a String
140      */
141     public String getLog(int log){
142
143       switch(log){
144
145       case EXCEPTION_LOG: return formatLog(exceptionLog);
146
147       case CONNECTION_LOG: return formatLog(connectionLog);
148
149           case DATA_PACKAGE_LOG: return formatLog(dataPackageLog);
150
151           case DEBUG_LOG: return formatLog(debugLog);
152
153       default: return "No such log";
154
155       }
156     }
157
158     /**
159      *
160      * This method extract the elements from the vector parameter
161      *
162      * @param log The Vector to extract elements from
163      * @return The contents of the Vector presented as a String
164      */
165     private String formatLog(Vector log){
166       String returnString = "";
167
168       // Runs through the vector and adds the elements to the returnString
169       for(int i = 0; i<log.size(); i++){
170         String element = (String)log.elementAt(i);
171         if(element != null) returnString += element+" \n----------\n ";
172         // = ""+element;
173       }
174       return returnString;
175     }
176
```

```
}
```

D.3.2 Class FileHandler

```
package peer2me.util;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.IOException;
import java.util.Enumeration;
import javax.microedition.io.Connector;
import javax.microedition.io.file.FileConnection;

/**
 * This class contains functionality for reading and writing all kinds of
 * files to and from the device file system.
 *
 * @author Torbjørn Vatn & Steinar A. Hestnes
 */
public class FileHandler{

  // The path to the file
  private String filePath;

  // The streams related to the file
  private DataInputStream inputStream;
  private DataOutputStream outputStream;

  // The connection to the file
  private FileConnection fileConnection;

  // How long to sleep from opening the stream to start reading/writing
  private final int sleepTime = 20;

  // The size of the blocks to read and write
  private int blockSize = 1024*50;

  // The total size of the file
  private long fileSize;

  // The number of written and read bytes
  private int readBytes;
  private int writtenBytes;

  // The byte[] temporary holding the bytes read and written to the file
  private byte[] readByteBlock;
```

```
44  private byte[] writeByteBlock;
45
46
47  /**
48   *
49   * Constructor.
50   *
51   * @param filePath The path to the file to be handled
52   */
53  public FileHandler(String filePath){
54    this.filePath = filePath;
55    this.readBytes = 0;
56    this.writtenBytes = 0;
57    this.fileSize = 0;
58  }
59
60
61  /**
62     *
63     * This method returns a list of the files in the given file path on the device
64     *
65     * @return A Enumeration containing the names of the files in the root directory
66     */
67  public Enumeration getFileList(){
68    Enumeration list = null;
69
70    try{
71      // Connects to the file
72      if(fileConnection == null){
73        fileConnection = (FileConnection) Connector.open("file:///"+filePath);
74        // Pauses the Thread for a while before using the fileConnection
75        try {
76          Thread.sleep(sleepTime);
77        } catch (InterruptedException ie) {
78          //This exception is irrelevant for the excecution
79        }
80      }
81      // Fetches the file list
82      list = fileConnection.list();
83      try {
84        // Pauses the Thread for a while before using the fileConnection
85        Thread.sleep(sleepTime);
86      } catch (InterruptedException ie) {
87        //This exception is irrelevant for the excecution
88      }
```

```
89
90      }catch(IOException ioe){
91        Log.getInstance().logException("FileHandler.getFileList()",ioe,false);
92      }
93
94      // Returns the list
95      return list;
96
97    }
98
99
100   /**
101     *
102     * This method returns the size of the this file
103     *
104     * @return The size as a long
105     */
106   public long getFileSize(){
107
108     try{
109       // Connects to the file
110       if(fileConnection == null){
111         fileConnection = (FileConnection) Connector.open("file:///"+filePath);
112         // Pauses the Thread for a while before using the fileConnection
113         try {
114           Thread.sleep(sleepTime);
115         } catch (InterruptedException ie) {
116           //This exception is irrelevant for the excecution
117         }
118       }
119       // Fetches the file size
120       fileSize = fileConnection.fileSize();
121       // Pauses the Thread for a while before using the fileConnection
122       try {
123         Thread.sleep(sleepTime);
124       } catch (InterruptedException ie) {
125         //This exception is irrelevant for the excecution
126       }
127     }catch(IOException ioe){
128       Log.getInstance().logException("FileHandler.FileHandler()",ioe,false);
129     }
130
131     // Returns the file size
132     return fileSize;
133   }
```

```
134
135     /**
136      *
137      * This method fetches the size of the blocks to read and write
138      *
139      * @return The blocksize
140      */
141     public int getBlockSize(){
142       return blockSize;
143     }
144
145     /**
146        *
147        * This method sets the size of the this file
148        *
149        * @param fileSize The size to set
150        */
151     public void setFileSize(long fileSize){
152       this.fileSize = fileSize;
153     }
154
155     /**
156      *
157      * This method reads the next byte in the file and returns it
158      *
159      * @return The next block of bytes
160      * @throws IOException This exception is thrown when the reading has failed
161      */
162     public synchronized byte[] readFile() throws IOException{
163
164       try{
165         // Connects to the file
166         if(fileConnection == null){
167           fileConnection = (FileConnection) Connector.open("file:///"+filePath);
168           // Pauses the Thread for a while before using the fileConnection
169           try {
170             Thread.sleep(sleepTime);
171           } catch (InterruptedException ie) {
172             //This exception is irrelevant for the excecution
173           }
174         }
175       }catch(IOException ioe){
176         Log.getInstance().logException("FileHandler.FileHandler()",ioe,false);
177       }
178
```

```
179     // Checks if the end of the file is reached
180     if (readBytes == fileConnection.fileSize ()) throw new EOFException();
181
182     try{
183       if (inputStream == null){
184         inputStream = fileConnection.openDataInputStream();
185         // Pauses the Thread for a while before using the inputStream
186         try {
187           Thread.sleep(sleepTime);
188         } catch (InterruptedException ie) {
189           //This exception is irrelevant for the excecution
190         }
191       }
192       // Reads the next block of bytes from the stream
193
194       // Checks if the file is smaller than the block size
195       if ( fileSize == 0)fileSize = fileConnection. fileSize ();
196       if ( fileSize < blockSize) blockSize = (int) fileSize ;
197
198       // Checks if the remaining unread bytes are less than block size
199       if ( fileSize - readBytes < blockSize){
200         blockSize = (int)( fileSize - readBytes);
201       }
202
203       readByteBlock = new byte[blockSize];
204       inputStream.read(readByteBlock, 0, blockSize);
205       readBytes += blockSize;
206       return readByteBlock;
207
208     }catch(IOException ioe){
209       closeFile ();
210       throw ioe;
211     }
212
213   }
214
215
216   /**
217    *
218    * This method writes the incoming byte block to the file
219    *
220    * @param theBytes The next byte block to write
221    * @param numberOfBytesRead The number of bytes in the theBytes[] array
222    * @throws IOException This exception is thrown when the writing has failed
223    *
```

```
224     */
225    public synchronized void writeFile(byte[] theBytes, int numberOfBytesRead) throws IOException{
226
227      // If input numberOfBytesRead == -1, the end of the file is reached
228      if(numberOfBytesRead == -1){
229        closeFile();
230        throw new EOFException();
231      }
232
233      try{
234        // Connects to the file
235        if(fileConnection == null){
236          fileConnection = (FileConnection) Connector.open("file:///"+filePath);
237          // Pauses the Thread for a while before using the fileConnection
238          try {
239            Thread.sleep(sleepTime);
240          } catch (InterruptedException ie) {
241            //This exception is irrelevant for the excecution
242          }
243
244          // Checks whether or not the file exists, if not create else delete and create
245          if(fileConnection.exists()){
246            // Deletes the existing file
247            fileConnection.delete();
248            // Creates a new file to write to
249            fileConnection.create();
250          }else{
251            // Creates a new file to write to
252            fileConnection.create();
253          }
254        }
255
256      }catch(IOException ioe){
257        Log.getInstance().logException("FileHandler.FileHandler()",ioe, false);
258      }
259
260      try{
261        if(outputStream == null){
262          outputStream = fileConnection.openDataOutputStream();
263          // Pauses the Thread for a while before using the outpuStream
264          try {
265            Thread.sleep(sleepTime);
266          } catch (InterruptedException ie) {
267            //This exception is irrelevant for the excecution
268          }
```

```
269         }
270
271         // Trims the last block of bytes
272         byte[] bytesToWrite = new byte[numberOfBytesRead];
273         // Copies the remaining bytes into a temporary byte[]
274         for(int i=0; i<numberOfBytesRead; i++){
275           bytesToWrite[i] = theBytes[i];
276         }
277         outputStream.write(bytesToWrite);
278
279       }catch(IOException ioe){
280         closeFile();
281         throw ioe;
282       }
283
284     }
285
286     /**
287      *
288      * This method writes the incoming byte to the file
289      *
290      * @param theByte The next byte to write
291      * @throws IOException This exception is thrown when the writing has failed
292      *
293      * @deprecated The method is substituted by writeFile(byte[] theBytes, int numberOfBytesRead)
294      */
295     public synchronized void writeFile(byte theByte) throws IOException{
296       try{
297         // Connects to the file
298         if(fileConnection == null){
299           fileConnection = (FileConnection) Connector.open("file:///"+filePath);
300           // Pauses the Thread for a while before using the fileConnection
301           try {
302             Thread.sleep(sleepTime);
303           } catch (InterruptedException ie) {
304             //This exception is irrelevant for the excecution
305           }
306
307           // Checks whether or not the file exists, if not create else delete and create
308           if(fileConnection.exists()){
309             // Deletes the existing file
310             fileConnection.delete();
311             // Creates a new file to write to
312             fileConnection.create();
313           }else{
```

```
314            // Creates a new file to write to
315            fileConnection.create();
316          }
317        }
318
319      }catch(IOException ioe){
320        Log.getInstance().logException("FileHandler.FileHandler()",ioe, false );
321      }
322
323      try{
324        if(outputStream == null){
325          outputStream = fileConnection.openDataOutputStream();
326          // Pauses the Thread for a while before using the outpuStream
327          try {
328            Thread.sleep(sleepTime);
329          } catch (InterruptedException ie) {
330            //This exception is irrelevant for the excecution
331          }
332        }
333
334        // Checks if the file size is less than the block size
335        if( fileSize < blockSize) blockSize = (int) fileSize ;
336
337        // Calculates the index to write the byte to
338        double numberOfBlocksWrittenDouble = (double)writtenBytes/(double)blockSize;
339        int numberOfBlocksWrittenInt = (int)(writtenBytes/blockSize);
340        int index = (int)((numberOfBlocksWrittenDouble-numberOfBlocksWrittenInt)*blockSize);
341
342
343
344        // Have to create a new writeByteBlock every time it is empty
345        if(writeByteBlock == null)writeByteBlock = new byte[blockSize];
346
347        writeByteBlock[index] = theByte;
348        writtenBytes++;
349
350        // Every time writeByteBlock is full it is written to file
351        if(index==(writeByteBlock.length-1) || writtenBytes == fileSize){
352          outputStream.write(writeByteBlock);
353          writeByteBlock = null;
354          // Checks if the remaining unread bytes are less than block size
355          if( fileSize - writtenBytes < blockSize){
356            writeByteBlock = new byte[((int)(fileSize - writtenBytes))];
357          }
358          if(writtenBytes == fileSize) throw new EOFException();
```

```
359         }
360
361       }catch(IOException ioe){
362         closeFile ();
363         throw ioe;
364       }
365     }
366
367     /**
368      *
369      * This method closes and nullifies the input- and ouput streams,
370      * and the file connection
371      *
372      */
373     public void closeFile (){
374       try{
375         // Pauses the Thread for a while before closing the streams and fileConnection
376         try {
377           Thread.sleep(sleepTime);
378         } catch (InterruptedException ie) {
379           //This exception is irrelevant for the excecution
380         }
381
382         if (inputStream != null)inputStream.close();
383         inputStream = null;
384         if (outputStream != null) outputStream.close();
385         outputStream = null;
386         if (fileConnection != null) fileConnection.close ();
387         fileConnection = null;
388       }catch (IOException ioe) {
389         Log.getInstance().logException("FileHandler.closeFile ()", ioe, false );
390       }
391     }
392
393
394 }
```

D.3.3 Class ASCIIToHexConvert

```
package peer2me.util;

/**
 *
 * This class converts ASCII letters into hexadecimal numbers
 *
 * @author Torbjørn Vatn & Steinar A. Hestnes
 */
public class ASCIIToHexConvert {

  /**
   * Constructor
   *
   */
  public ASCIIToHexConvert(){
  }

  /**
   *
   * This method returns a String with hex representations of each character in the provided String
   *
   * @param ascii The String to convert
   * @return A String with hex representations
   */
  public String convertASCIIToHex(String ascii){
    String hexString = "";

    for(int i=0; i<ascii.length(); i++){
      hexString += findHexValue(ascii.charAt(i));
    }
    return hexString;
  }

  /**
   *
   * This method has a switch case structure that contains the hex values for the symbols:
   * 0-9, A-Z and a-z
   *
   * @param input The char to convert to hex
   * @return The retrived hex value, "00" is returned in cases where no value is found in the switch
   */
  private String findHexValue(char input){

```

```
    String hex = "";

    switch(input){
      case '0' : hex = "30"; break; case 'V' : hex = "56"; break;
      case '1' : hex = "31"; break; case 'W' : hex = "57"; break;
      case '2' : hex = "32"; break; case 'X' : hex = "58"; break;
      case '3' : hex = "33"; break; case 'Y' : hex = "59"; break;
      case '4' : hex = "34"; break; case 'Z' : hex = "5A"; break;
      case '5' : hex = "35"; break; case 'a' : hex = "61"; break;
      case '6' : hex = "36"; break; case 'b' : hex = "62"; break;
      case '7' : hex = "37"; break; case 'c' : hex = "63"; break;
      case '8' : hex = "38"; break; case 'd' : hex = "64"; break;
      case '9' : hex = "39"; break; case 'e' : hex = "65"; break;
      case 'A' : hex = "41"; break; case 'f' : hex = "66"; break;
      case 'B' : hex = "42"; break; case 'g' : hex = "67"; break;
      case 'C' : hex = "43"; break; case 'h' : hex = "68"; break;
      case 'D' : hex = "44"; break; case 'i' : hex = "69"; break;
      case 'E' : hex = "45"; break; case 'j' : hex = "6A"; break;
      case 'F' : hex = "46"; break; case 'k' : hex = "6B"; break;
      case 'G' : hex = "47"; break; case 'l' : hex = "6C"; break;
      case 'H' : hex = "48"; break; case 'm' : hex = "6D"; break;
      case 'I' : hex = "49"; break; case 'n' : hex = "6E"; break;
      case 'J' : hex = "4A"; break; case 'o' : hex = "6F"; break;
      case 'K' : hex = "4B"; break; case 'p' : hex = "70"; break;
      case 'L' : hex = "4C"; break; case 'q' : hex = "71"; break;
      case 'M' : hex = "4D"; break; case 'r' : hex = "72"; break;
      case 'N' : hex = "4E"; break; case 's' : hex = "73"; break;
      case 'O' : hex = "4F"; break; case 't' : hex = "74"; break;
      case 'P' : hex = "50"; break; case 'u' : hex = "75"; break;
      case 'Q' : hex = "51"; break; case 'v' : hex = "76"; break;
      case 'R' : hex = "52"; break; case 'w' : hex = "77"; break;
      case 'S' : hex = "53"; break; case 'x' : hex = "78"; break;
      case 'T' : hex = "54"; break; case 'y' : hex = "79"; break;
      case 'U' : hex = "55"; break; case 'z' : hex = "7A"; break;

      default:
        hex = "00";

    }

    // the retrived hex value
    return hex;
  }
}
```

D.4 Package peer2me.network

D.4.1 Class Network

```
package peer2me.network;

import java.io.IOException;

import peer2me.util.Log;
import peer2me.domain.DataPackage;
import peer2me.framework.FrameworkFrontEnd;

/**
 * This is the super class of the technology specific network classes. Methods
 * that are equal for all the sub classes are located in this super class, and there are
 * abstact methods that the sub classes have to implement. The getInstance() method
 * in this class returns a reference to the preferred network sub class.
 *
 * @author Torbjørn Vatn & Steinar A. Hestnes
 */
public abstract class Network{

    // The Log instance
    private static Log log;

  // The Network instance returned by the getInstance() method
  private static Network singleton;

    // A reference to the FrameworkFrontEnd
    private FrameworkFrontEnd frameworkFrontEnd;

    // The applicationId of the MIDlet
    private String applicationID;

    // The ConnectionListener of the Network
    private ConnectionListener connectionListener;

  /**
   *
   * This method returns an instance of the preferred network.
   * It is called from FrameworkFrontEnd.initFramework().
   *
   * @param preferredNetwork Indicating which network implementation to use.
   * @throws ClassNotFoundException The input preferredNetwork is invalid
   * @throws IllegalAccessException The input preferredNetwork is invalid
```

```
42     * @throws InstantiationException The input preferredNetwork is invalid
43     * @return The Network instance
44     */
45    public static synchronized Network getInstance(String preferredNetwork) throws ClassNotFoundException,
46                                                                    IllegalAccessException, InstantiationException{
47
48      // A log instance
49      log = Log.getInstance();
50
51      if(singleton != null){
52        return singleton;
53      }else{
54        try{
55          // Fetching a instance of the preferred network class
56          singleton = (Network)Class.forName(preferredNetwork).newInstance();
57
58        }catch(ClassNotFoundException cnfe){
59          log.logException("Network.getInstance()",cnfe, false);
60          throw cnfe;
61        }catch(IllegalAccessException iae){
62          log.logException("Network.getInstance()",iae, false);
63          throw iae;
64        }catch(InstantiationException ie){
65          log.logException("Network.getInstance()",ie, false);
66          throw ie;
67        }
68
69        // Returning the singleton instance
70        return singleton;
71      }
72    }
73
74
75      /**
76       *
77       * This method returns a reference to the instance of the preferred network.
78       * It is used when an instance already is created and a reference to
79       * this instance is needed.
80     *
81     * @throws ClassNotFoundException The input preferredNetwork is invalid
82     * @throws IllegalAccessException The input preferredNetwork is invalid
83     * @throws InstantiationException The input preferredNetwork is invalid
84     * @return The Network instance
85       */
86      public static synchronized Network getInstance(){
```

```
87          // Checks whether the other getInstance method has already been run
88          if(singleton==null){
89              log.logDebugInfo("Network.getInstance()","No Network instance found!");
90              return null;
91          }
92          return singleton;
93      }
94
95
96      /**
97       *
98       * This method sets a reference to the ConnectionListener
99       *
100      * @param connectionListener A reference to the ConnectionListener
101      */
102     public void setConnectionListener(ConnectionListener connectionListener){
103         this.connectionListener = connectionListener;
104     }
105
106         /**
107          *
108          * This method returns the ConnectionListener reference
109          *
110          * @return connectionListener A reference to the ConnectionListener
111          */
112         public ConnectionListener getConnectionListener(){
113             return connectionListener;
114         }
115
116         /**
117          *
118          * This method sets a reference to the FrameworkFrontEnd
119          *
120          * @param frameworkFrontEnd A reference to the FrameworkFrontEnd
121          */
122         public void setFrameworkFrontEnd(FrameworkFrontEnd frameworkFrontEnd){
123             this.frameworkFrontEnd = frameworkFrontEnd;
124         }
125
126         /**
127          *
128          * This method returns the FrameworkFrontEnd reference
129          *
130          * @return frameworkFrontEnd A reference to the FrameworkFrontEnd
131          */
```

```
132     public FrameworkFrontEnd getFrameworkFrontEnd(){
133         return frameworkFrontEnd;
134     }
135
136     /**
137      *
138      * This method sets the applicationID.
139      * The application ID must be used to ensure that all nodes joining
140      * the network are running the same MIDlet.
141      *
142      * @param applicationID The ID of the MIDlet (e.g. the MIDlet name)
143      */
144   public void setApplicationId(String applicationID){
145     this.applicationID = applicationID;
146   }
147
148   /**
149    *
150    * This method returns the applicationId
151    *
152    * @return applicationID The ID of the MIDlet
153    */
154   public String getApplicationId(){
155     return applicationID;
156   }
157
158
159   /*************************************************************************/
160   // The abstract methods inherited and overridden by the sub network classes
161   /*************************************************************************/
162
163   /**
164    * Initiates the network instance.
165    * It is called from the FrameworkFrontEnd.initFramework()
166    *
167    * @throws Exception Failed to initiate the network
168    */
169   public abstract void init () throws Exception;
170
171   /**
172    * Starts a search for devices running the same MIDlet
173    *
174    * @throws IOException Error during the search
175    */
176   public abstract void searchForNodes() throws IOException;
```

```
177
178     /**
179      *
180      * Called when the same MIDlet is found on a remote device
181      *
182      * @param input An object representing the connection to the found node.
183      */
184     public abstract void nodeFound(Object input) throws IOException;
185
186     /**
187      *
188      * This method fetches the name of the remote node.
189      *
190      * @param input An object representing the connection to the found node.
191      * @return The name of the remote node.
192      */
193     public abstract String getRemoteNodeName(Object input);
194
195     /**
196      *
197      * This method establishes a connection to the chosen node.
198      * It could e.g. be run from the Network.sendDataPackage() to connect
199      * before sending a package.
200      *
201      * @param nodeAddress The address to the node to connect to
202      *
203      */
204     public abstract void connectToNode(String nodeAddress);
205
206     /**
207      *
208      * This method is called from the ConnectionListener.run() when
209      * the acceptAndOpen() method in ConnectionListener.run() is done.
210      *
211      */
212     public abstract void connectionEstablished();
213
214     /**
215      *
216      * This method returns the node address.
217      *
218      * @param input String "localNode" to retreive the address of the local device.
219      * A object representing the connection to the remote node to retreive the
220      * address of a remote device.
221      *
```

```
222      * @return The node network address.
223      * @throws IOException
224      *
225      */
226     public abstract String getNodeAddress(Object input) throws IOException;
227
228
229     /**
230      *
231      * This method is used by the FrameworkFrontEnd to send a data package of
232      * any sort to a remote node.
233      *
234      * @param dataPackage The data package to be sent
235      * @param recipients A list containing addresses to the recipient nodes
236      *
237      */
238     public abstract void sendDataPackage(DataPackage dataPackage, String[] recipients);
239
240
241 }
```

D.4.2 Class NodeConnection

```
package peer2me.network;

import peer2me.util.FileHandler;
import peer2me.util.Log;
import peer2me.domain.DataPackage;
import peer2me.domain.FilePackage;
import peer2me.domain.GroupSyncPackage;
import peer2me.domain.Node;
import peer2me.domain.TextPackage;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.IOException;
import java.util.Date;
import java.util.Vector;
import javax.microedition.io.StreamConnection;


/**
 *
 * This class contains a thread that runs on each connected node and listens for
 * incoming data packages and sends data packages out.
 * It is created and started in NodeConnection.startNodeConnection().
 *
 * @author Torbjørn Vatn & Steinar A. Hestnes
 */
public class NodeConnection{

    // The Log instance
    private Log log;

    // The Network instance of the preferred network
    private Network currentNetwork;

    // The different variables concerning the in- and out streams
    private DataInputStream inputStream;
    private DataOutputStream outputStream;
    private StreamConnection connection;
    private Node node;

    // The queue holding the data packages to send
    private Vector sendQueue;

```

```
44      // Boolean values that controls whether or not the Input- and OutputStreams are allowed to perform their tasks
45      private boolean openInputStream;
46      private boolean openOutputStream;
47
48      // The threads running whenever a connection is open.
49      private InputThread inputThread;
50      private OutputThread outputThread;
51
52      /**
53       *
54       * Constructor.
55       * This constructor is called from the constructor in class Node.
56       *
57       * @param connection The connection to the node
58       * @param node The node that owns this NodeConnection
59       */
60      public NodeConnection(StreamConnection connection, Node node){
61
62          // Fetches a instance of the Log
63          log = Log.getInstance();
64          // Sets the connection to connect to and fetches an instance of the currentNetwork
65          this.connection = connection;
66          this.node = node;
67          currentNetwork = Network.getInstance();
68          // Creates the sendQue
69          sendQueue = new Vector();
70
71          // The Input- and OutputStreams shall not do anything before the node is connected
72          // These values are toggled from ConnectionListener.run() and NodeConnection.sendDataPackage()
73          openInputStream = false;
74          openOutputStream = false;
75          // Starts a thread that processes the sendQue
76          outputThread = new OutputThread();
77          outputThread.setPriority(Thread.MAX_PRIORITY);
78          outputThread.start();
79          // Starts a thread constantly listening for incoming datapackages
80          inputThread = new InputThread();
81          inputThread.setPriority(Thread.MAX_PRIORITY);
82          inputThread.start();
83      }
84
85      /**
86       *
87       * This method return the size of the sendQue.
88       *
```

```
 * @return The size of the sendQue
 */
public int getSendQueueSize(){
  return sendQueue.size();
}


/**
 *
 * This method receives incoming datapackages from remote nodes.
 * It is called in an infinite loop in the private class InputThread
 * in this class.
 *
 */
public void processIncomingData(){
    if(connection != null){
      boolean connectionFailed = false;
        try{
            if(inputStream == null){
                inputStream = connection.openDataInputStream();
            }
        }catch(IOException ioe1){
            connectionFailed = true;
            log.logException("NodeConnection.processIncomingData()1",ioe1,true);
            // Opening of streams failed, ergo connection lost
            // Close connection and inform the NodeListener
            try {
                connection.close();
                // The connection must be set to null to stop the thread running
                // this method when the connection has closed
                connection = null;
            } catch (IOException ioe2){
                log.logException("NodeConnection.processIncomingData()2",ioe2,true);
            }
        }

        // If an inputstream and an outputstream was successfully opened, a infinite loop starts
        if(!connectionFailed){
            try {
                while(inputStream != null && connection != null && !connectionFailed){
                    int type = -1;
                    try{
                        // Reads the type of the data package
                        type = inputStream.readInt();
                    }catch(IOException ioe){
```

```
134                     connectionFailed = true;
135                 }
136                 // The type of the package determines what should be done with the package
137                 switch(type){
138
139                 case(DataPackage.GROUP_SYNC_PACKAGE):
140                     // Reads the length of the incoming package
141                     int byteLength1 = inputStream.readInt();
142                     byte[] bytes1 = new byte[byteLength1];
143                     // Reads the incoming bytes
144                     for(int i=0;i<bytes1.length;i++){
145                       bytes1[i] = inputStream.readByte();
146                     }
147
148                     // Checks if the sendQue on the sender side is empty
149                     if(inputStream.readBoolean()){
150                       // Closes the connection if the remote node is finished sending all its datapackages
151                       openInputStream = false;
152                     }
153
154                     GroupSyncPackage groupSyncPackage = new GroupSyncPackage();
155
156                     // Interprets the content and sets the variables in the groupSyncPackage object
157                     groupSyncPackage.parseBytes(bytes1);
158
159                     // Notifies the midlet via the frontEnd about the received message.
160                     currentNetwork.getFrameworkFrontEnd().notifyAboutReceivedGroupSyncPackage(groupSyncPackage);
161                     break;
162
163                 case(DataPackage.TEXT_PACKAGE):
164                     // Reads the length of the incoming package
165                     int byteLength2 = inputStream.readInt();
166                     byte[] bytes2 = new byte[byteLength2];
167
168                     // Reads the incoming bytes in blocks
169                     // Reading blocks increases the transfer rate considerably
170                     boolean finishedReading = false;
171                 int blockSize = 200;
172                 int totalRead = 0;
173                 while(!finishedReading){
174                   // If whats left is less than one blockSize
175                   if(byteLength2 - totalRead < blockSize) blockSize = byteLength2-totalRead;
176                   byte[] block = new byte[blockSize];
177                   int numberRead = inputStream.read(block,0,blockSize);
178                   // Stores whats read in an array large enough for the whole package
```

```
179                 for(int i=0; i<numberRead; i++){
180                   bytes2[totalRead] = block[i];
181                   totalRead++;
182                 }
183                 if(totalRead == byteLength2) finishedReading = true;
184               }
185
186                 // Checks if the sendQue on the sender side is empty
187                 if(inputStream.readBoolean()){
188                   // Closes the connection if the remote node is finished sending all its datapackages
189                   openInputStream = false;
190                 }
191
192                 // Notifies the midlet via the frontEnd about the received message.
193                 TextPackage textPackage = new TextPackage();
194                 textPackage.parseBytes(bytes2);
195
196                 currentNetwork.getFrameworkFrontEnd().notifyAboutReceivedTextPackage(textPackage);
197                 break;
198
199               case(DataPackage.FILE_PACKAGE):
200                 // Reads the length of the incoming package
201                 int byteLength3 = inputStream.readInt();
202                 byte[] bytes3 = new byte[byteLength3];
203                 // Reads the incoming bytes
204                 for(int i=0;i<bytes3.length;i++){
205                     bytes3[i] = inputStream.readByte();
206                 }
207
208                 // Creates a filePackage based on the received data
209                 FilePackage filePackage = new FilePackage();
210                 filePackage.parseBytes(bytes3);
211                 // Reads the file and writes it to the filesystem
212                 FileHandler fileHandler = new FileHandler(filePackage.getFilePath());
213                 // Fetches the size of the file and sets it in the fileHandler
214                 fileHandler.setFileSize(filePackage.getFileSize());
215
216                 // Checks if the sendQue on the sender side is empty
217                 if(inputStream.readBoolean()){
218                   // Closes the connection if the remote node is finished sending all its datapackages
219                   openInputStream = false;
220                 }
221
222                 boolean endOfFile= false;
223                 while(!endOfFile){
```

```
224                                 try{
225                                     byte[] theBytes = new byte[fileHandler.getBlockSize()];
226                                     // Reads data from the inputStream into a byte table
227                                     int numberOfBytesRead = inputStream.read(theBytes, 0, fileHandler.getBlockSize());
228                                     // Writes the bytes to file
229                                 fileHandler.writeFile(theBytes, numberOfBytesRead);
230
231                                 }catch(EOFException eofe){
232                                     endOfFile = true;
233                                     fileHandler.closeFile();
234                                 }
235                             }
236
237                             currentNetwork.getFrameworkFrontEnd().notifyAboutReceivedFilePackage(filePackage);
238                             break;
239
240                         default:
241                             break;
242                         }
243                     }
244
245                 }catch(IOException ioe) {
246                     log.logException("NodeConnection.processIncomingData()3", ioe, true);
247                 }
248             }
249         }
250     }
251
252
253
254     /**
255      *
256      * This method sends datapackages to remote nodes.
257      * It processes the que of unsent datapackages.
258      * It is called in an infinite loop in the private class OutputThread
259      * in this class.
260      *
261      */
262     public synchronized void processSendQueue(){
263         if(connection != null){
264         if(sendQueue.size() > 0){
265           // Retriving the data packages to send from the sendQue
266           DataPackage dataPackage = (DataPackage)sendQueue.firstElement();
267           sendQueue.removeElement(dataPackage);
268           // A byte table holding the data to send
```

```
269         byte[] data = dataPackage.toSendableFormat();
270
271         try{
272           // Opening the output stream if it is not allready open
273           if(outputStream == null){
274                   outputStream = connection.openDataOutputStream();
275               }
276
277           // Saves a timestamp used to estimate the transfer rate
278           long startTime = new Date().getTime();
279
280           // Sending the type of the data package over the steam
281               outputStream.writeInt(dataPackage.getType());
282
283               // Sending the length of the data package over the steam
284           outputStream.writeInt(data.length);
285
286           // Sends the data package in blocks over the stream
287               // Sending blocks instead of single bytes increases the transfer rate considerably
288           boolean finishedWriting = false;
289           int blockSize = 200;
290           int totalWritten = 0;
291           while(!finishedWriting){
292             // If whats left is less than one blockSize
293             if(data.length - totalWritten < blockSize) blockSize = data.length-totalWritten;
294             byte[] block = new byte[blockSize];
295             // Fills the byte array to be sent
296             for(int i=0; i<blockSize; i++){
297               block[i] = data[totalWritten];
298               totalWritten++;
299             }
300             outputStream.write(block);
301
302             if(totalWritten == data.length) finishedWriting = true;
303           }
304
305               // If the datapackage is a FilePackage we have to send the content
306               // of the file
307               long fileSize = 0;
308               if(dataPackage.getType() == DataPackage.FILE_PACKAGE){
309                 // Opens the file handler
310                 FileHandler fileHandler = new FileHandler(((FilePackage)dataPackage).getFilePath());
311
312                 // Flushes the output stream
313                   outputStream.flush();
```

```
314
315                 boolean endOfFile = false;
316                 while(!endOfFile){
317                   try{
318                     byte[] theBytes = fileHandler.readFile();
319                     outputStream.write(theBytes);
320                   }catch(EOFException eofe){
321                     endOfFile = true;
322                     fileHandler.closeFile();
323                   }
324                 }
325                 fileSize = ((FilePackage)dataPackage).getFileSize();
326               }
327
328
329               // Logs a message if the text package was sent successfully
330               if(dataPackage.getType() == DataPackage.TEXT_PACKAGE ||
331                     dataPackage.getType() == DataPackage.FILE_PACKAGE){
332                 // Estimates the transfer rate of the file
333                 long endTime = new Date().getTime();
334                 long transferTime = (endTime-startTime)/1000;
335                 if(transferTime==0) transferTime = 1;
336                 double kBps = ((double)(data.length+fileSize)/1024)/(double)transferTime;
337
338                 //the code below calculates and rounds off the transfer rate with three decimals
339                 String rate = Double.toString(kBps);
340                 int commaIndex = rate.indexOf(".");
341                 int decimal3 = Integer.parseInt(""+rate.charAt(commaIndex+3)+"");
342                 int decimal4 = Integer.parseInt(""+rate.charAt(commaIndex+4)+"");
343                 rate = rate.substring(0,commaIndex+4);
344                 if(decimal4>=5){
345                     if(decimal3 == 9){
346                         decimal3 = decimal3+1;
347                         rate = rate.substring(0,commaIndex+2);
348                         rate += decimal3;
349                     }
350                     else{
351                         decimal3 = decimal3+1;
352                         rate = rate.substring(0,commaIndex+3);
353                         rate += decimal3;
354                     }
355                 }
356
357               log.logDataPackage("Finished transfering data to "+
358                         node.getNodeName()+". (Transfer rate was "+rate+"kB/s)");
```

```
359                     }
360
361             }catch(IOException ioe){
362               // Because this method is called from within a run() the log has to noitfy the MIDLet of the exception
363               log.logException("NodeConnection.processSendQue()",ioe,true);
364               closeConnection();
365               // Tries to send the datapackage once more
366               currentNetwork.sendDataPackage(dataPackage,dataPackage.getRecipients());
367           }
368           }
369
370
371           // If the queue is not empty, the processing continues
372           try {
373             Thread.sleep(500);
374         } catch (InterruptedException ie) {
375           // do nothing
376         }
377           try{
378             // Closes the outputstream if the sendQueue is empty
379                   // The connections are re-established when a new datapackage is sent
380                   if(sendQueue.size() == 0){
381                     // Notifies the remote recipient that we are closing the stream
382                     outputStream.writeBoolean(true);
383                     openOutputStream = false;
384                     // Flushes the output stream
385                     outputStream.flush();
386                   }else{
387                      outputStream.writeBoolean(false);
388                      // Flushes the output stream
389                      outputStream.flush();
390                      // Must process the next package
391                    processSendQueue();
392                   }
393         }catch(IOException ioe){
394               // Because this method is called from within a run() the log has to noitfy the MIDLet of the exception
395               log.logException("NodeConnection.processSendQue()2",ioe,true);
396               closeConnection();
397           }
398           }
399       }
400
401
402       /**
403        *
```

```
404     * This method is called by the sendMessage() method in the Network class
405     * when a data package is sent to the Node associated with this
406     * NodeConnection.
407     *
408     * @param dataPackage The DataPackage to send
409     */
410    public synchronized void sendDataPackage(DataPackage dataPackage){
411      sendQueue.addElement(dataPackage);
412        // Continues to run the Input- and OutputStreams on the
413        // representation of the remote recipient node
414        openOutputStream();
415    }
416
417
418    /**
419     *
420     * This method returns the connection object.
421     *
422     * @return An object representing the connection to the remote node
423     */
424    public StreamConnection getConnection(){
425        return connection;
426    }
427
428    /**
429     *
430     * This method updates the connection object. It is used when the existing
431     * connection is closed and a new open connection is needed.
432     *
433     * @param connection The connection to the remote node
434     */
435    public void setConnection(StreamConnection connection){
436        this.connection = connection;
437    }
438
439    /**
440     *
441     * This method sets a boolean that controls whether or not the InputStream
442     * are allowed to listen for incoming data.
443     * The value is toggled from ConnectionListener.run().
444     *
445     */
446    public void openInputStream(){
447      this.openInputStream = true;
448      // Starts the steams again
```

```
449         inputThread.restartThread();
450     }
451
452     /**
453      *
454      * This method sets a boolean that controls whether or not the OutputStream
455      * are allowed to send data.
456      * The value is toggled from NodeConnection.sendDataPackage()
457      *
458      */
459     public void openOutputStream(){
460         this.openOutputStream = true;
461         // Starts the steams again
462         outputThread.restartThread();
463     }
464
465     /**
466      *
467      * This method closes the input- and output streams and the connection.
468      * It is called from Group.shutdownGroup() to clean up during shutdown.
469      *
470      */
471     public void closeConnection(){
472         try{
473             if(outputStream != null) this.outputStream.close();
474             this.outputStream = null;
475             if(inputStream != null) this.inputStream.close();
476             this.inputStream = null;
477             if(connection != null) this.connection.close();
478             this.connection = null;
479     }catch(IOException ioe){
480         // Because this method is called from within a run() the log has to noitfy the MIDLet of the exception
481         log.logException("NodeConnection.closeConnection()",ioe,true);
482     }
483     }
484
485
486     /**
487      *
488      * This class is a thread listening for incoming datapackages from remote nodes.
489      * The thread is started from the constructor in class NodeConnection.
490      *
491      * @author Torbjørn Vatn & Steinar A. Hestnes
492      */
493     private class InputThread extends Thread{
```

```
494
495         /**
496          * Constantly running. It is run from the constructor in class NodeConnection.
497          * Constantly listening for incoming datapackages.
498          *
499          */
500         public void run(){
501             while(true){
502                 if(openInputStream){
503                   try{
504                     // Sleeps the thread to ensure that the connection is open
505                       sleep(500);
506                   }catch(InterruptedException ie){
507                       // This exception is irrelevant for the excecution
508                   }
509                 // Calls the processIncomingData() method to receive data packages from remote nodes
510                   processIncomingData();
511
512                   if(!openInputStream){
513                     try{
514                     // Sleeps the thread to ensure that the connection is ready to be closed
515                     sleep(500);
516                     }catch(InterruptedException ie){
517                         // This exception is irrelevant for the excecution
518                     }
519                     closeConnection();
520                   }
521                 }
522                 // Pauses the thread until it is notified again
523                 this.pauseThread();
524             }
525         }
526
527         /**
528          * This method pauses this thread
529          *
530          */
531         private synchronized void pauseThread(){
532           try{
533             this.wait();
534           }catch(InterruptedException ie){
535             // This exception is irrelevant for the excecution
536           }
537         }
538
```

```
539         /**
540          * This method notifies this thread
541          *
542          */
543         public synchronized void restartThread(){
544           this.notify();
545         }
546     }
547
548
549     /**
550      *
551      * This class is a thread sending datapackages to remote nodes.
552      * The thread is started from the constructor in class NodeConnection.
553      *
554      * @author Torbjørn Vatn & Steinar A. Hestnes
555      */
556     private class OutputThread extends Thread{
557
558         /**
559          * Constantly running. It is run from the constructor in class NodeConnection.
560          * Processes the sendQue.
561          *
562          */
563         public void run(){
564             while(true){
565               if(openOutputStream){
566                 try{
567                   // Sleeps the thread to ensure that the connection is open
568                   sleep(500);
569                   }catch(InterruptedException ie){
570                       // This exception is irrelevant for the excecution
571                   }
572                   // Calls the processSendQue() method to send any unsendt data packages
573                   processSendQueue();
574
575                   if(!openOutputStream){
576                     try{
577                     // Sleeps the thread to ensure that the connection is ready to be closed
578                     sleep(100);
579                     }catch(InterruptedException ie){
580                         // This exception is irrelevant for the excecution
581                     }
582                     closeConnection();
583                   }
```

```
584                 }
585                 // Pauses the thread until it is notified again
586                   this.pauseThread();
587               }
588           }
589
590
591           /**
592            * This method pauses this thread
593            *
594            */
595           private synchronized void pauseThread(){
596             try{
597               this.wait();
598             }catch(InterruptedException ie){
599               // This exception is irrelevant for the excecution
600             }
601           }
602
603           /**
604            * This method notifies this thread
605            *
606            */
607           public synchronized void restartThread(){
608             this.notify();
609           }
610       }
611
612
613   }
```

D.4.3 Class ConnectionListener

```
package peer2me.network;

import peer2me.util.Log;
import peer2me.domain.Node;
import java.io.IOException;
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import javax.microedition.io.StreamConnectionNotifier;


/**
 * This class contains a ConnectionListener thread listening for
 * incoming connection attempts from other devices running
 * the same MIDlet built upon the framework.
 * When a incomming connection is detected, a Node representation is created
 * representing the connecting device.
 * A ConnectionListener thread is created in Network.init().
 *
 * @author Torbjørn Vatn & Steinar A. Hestnes
 */
public class ConnectionListener implements Runnable{

  // The Log instance
  private Log log;

  // The network instance related to this ConnectionService
  private Network currentNetwork;
  // The ConnectionURL that the ConnectionService should listen to
  private String connectionURL;
  // The StreamConnectionNotifier that creates a StreamConnection that
  // represents a server side socket connection
  private StreamConnectionNotifier connectionNotifier = null;
  // The StreamConnection representing the open connection
  private StreamConnection connection = null;
  // This ConnectionService thread
  private Thread localThread;
  // Whether or not the Thread should shut down
  private boolean shutdown = false;
  // Whether or not the connection has failed
  private boolean failed = false;

  /**
   * Constructor.
```

```
44     * A ConnectionListener is created in the Network.init() method.
45     *
46     * @param connectionURL The ConnectionURL to listen to
47     */
48    public ConnectionListener(String connectionURL){
49
50      this.currentNetwork = Network.getInstance();
51      this.connectionURL = connectionURL;
52
53      // Fetches a instance of the Log
54      log = Log.getInstance();
55
56      // Starts a ConnectionListener thread listening for a connection
57      localThread = new Thread(this);
58      localThread.start();
59    }
60
61    /**
62     * This method is called when the ConnectionListener thread is started in the
63      * constructor.
64      *
65      * It continously listens for incoming connections matching the
66      * serviceID of the peer2me framework. The listener is "passive" and opens a
67      * connection waiting for a device to take contact.
68      * If an incoming connetion occurs, information is abstracted from the remote
69      * node, and a node object containing this connection is created and added
70      * to the group on the local node.
71      *
72     */
73    public void run(){
74      try{
75        // Opens the stream
76        connectionNotifier = (StreamConnectionNotifier) Connector.open(connectionURL);
77
78        while(true){//(!failed || !shutdown){
79
80                // Returns a StreamConnection that represents a server side socket connection.
81          connection = (StreamConnection)connectionNotifier.acceptAndOpen();
82
83          // If the connection is successful a representation of the remote node is created
84          if (!failed || !shutdown){
85
86              // Creates the node who found "me" (as a participant) and opens a connection
87                      String address = Network.getInstance().getNodeAddress(connection);
88
```

```
                    Node remoteNode = new Node(address,connection);

                    // Starts the InputThread in NodeConnection again since we have a connection
                    remoteNode.getNodeConnection().openInputStream();

                    // Adds the node who found "me" to the group containing all found nodes
                    currentNetwork.getFrameworkFrontEnd().getGroup().addParticipant(remoteNode);

                    // This Node is discovered and connected by another Node the currentNetwork is notified
                    currentNetwork.connectionEstablished();

                    log.logConnection("Connected successfully to a node with address: "+address);
                }
            }

        }catch(IOException ioe) {
            log.logException("ConnectionListener.run()",ioe, true);
            failed = true;
        }catch(SecurityException se) {
            log.logException("ConnectionListener.run()",se, true);
            failed = true;
        }catch(IllegalArgumentException iae){
            log.logException("ConnectionListener.run()",iae, true);
            failed = true;
        }
    }

    /**
     *
     * This method shuts down this thread and closes the connection to clean up.
     * It is called from FrameworkFrontEnd.shutdownFramework().
     *
     */
    public void shutdown(){
        // Have to shut down
        shutdown = true;
        // Closes the connection to clean
        try{
            connectionNotifier.close();
        }catch (IOException ioe) {
            log.logException("ConnectionListener.shutdown()",ioe,false);
        }
    }


```

}

D.5 Package peer2me.network.bluetooth

D.5.1 Class BluetoothNetwork

```
package peer2me.network.bluetooth;

import java.io.IOException;
import java.util.Hashtable;
import java.util.Vector;

import javax.bluetooth.BluetoothStateException;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.RemoteDevice;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.DataElement;
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import peer2me.util.ASCIIToHexConvert;
import peer2me.util.Log;
import peer2me.network.ConnectionListener;
import peer2me.network.Network;
import peer2me.network.NodeConnection;
import peer2me.domain.DataPackage;
import peer2me.framework.FrameworkFrontEnd;


/**
 * This class is a bluetooth specific sub class of the Network class
 * and implements all the abstract methods of it's parent class in a bluetooth
 * context. It uses the bluetooth Java API, JSR-82, to perform operations on
 * the bluetooth hardware of the mobile device.
 *
 * @author Torbjørn Vatn & Steinar A. Hestnes
 */
public class BluetoothNetwork extends Network implements BluetoothServiceDiscoveryListener{

    // The Log instance
    private Log log;

    // The connectionURL used by the ConnectionService created in the init() methood
  private String connectionURL;

  // The UUID number generated using the tool found at http://kruithof.xs4all.nl/uuid/uuidgen
  private String generatedUuid = "0ade9c80bb2b11daa94d0800200c9a66";

```

```
42      // A reference to BluetoothServiceDiscovery
43      private BluetoothServiceDiscovery bluetoothServiceDiscovery;
44
45      // A list containing addresses and serviceRecords of all found nodes
46      // (nodes who are running the same application but not connected yet)
47      private Hashtable foundNodes;
48
49      // This boolean decides whether or not this Node is connected to another node.
50      private boolean isConnected;
51
52      // A boolean indicating whether the serviceSearch is completed or not
53      private boolean serviceSearchCompleted;
54
55      // A boolean indicating whether the serviceSearch failed or not
56      private boolean serviceSearchFailed;
57
58
59    /**
60     *
61     * Constructor. Protected to ensure singleton pattern.
62     *
63     */
64    public BluetoothNetwork(){
65          // Fetches a instance of the Log
66          log = Log.getInstance();
67      }
68
69    /**
70     * Initiates the network instance.
71     * It is called from the FrameworkFrontEnd.initFramework()
72     *
73     * @throws BluetoothStateException Failed to initiate the network
74     */
75    public void init() throws BluetoothStateException{
76
77      isConnected = false;
78      serviceSearchCompleted = false;
79      serviceSearchFailed = false;
80
81
82      // Sets the connectionURL used by the ConnectionListener
83          String localNodeName = getFrameworkFrontEnd().getLocalNode().getNodeName();
84          connectionURL = "btspp://localhost:"+getUUIDString()+";authenticate=false;encrypt=false;name="+localNodeName;
85
86      // Have to set the local device discoverable
```

```
87          try {
88              LocalDevice.getLocalDevice().setDiscoverable(javax.bluetooth.DiscoveryAgent.GIAC);
89          } catch (BluetoothStateException bse) {
90              log.logException("ConnectionListener.ConnectionListener()",bse,false );
91              throw bse;
92          }
93
94          foundNodes = new Hashtable();
95          // Creates the class that contains low level Bluetooth discovery operations.
96          bluetoothServiceDiscovery = new BluetoothServiceDiscovery();
97
98      /* The ConnectionListener instance that listens for incoming requests from
99       * other nodes in discovery mode. When this node is discovered the "discoverer"
100      * can choose to create a connection between the two, and the remote node is
101      * represented by a node object localy on this node.
102      */
103         setConnectionListener(new ConnectionListener(connectionURL));
104     }
105
106
107     /**
108      *
109      * This method is called from the ConnectionListener.run() when
110      * the acceptAndOpen() method in ConnectionListener.run() is done.
111      *
112      */
113     public void connectionEstablished(){
114         // Indicates that this node was connected (contacted) by another node
115         isConnected = true;
116     }
117
118     /**
119      * Starts a search for devices running the same MIDlet
120      *
121      * @throws IOException Error during the search
122      *
123      */
124     public void searchForNodes() throws IOException{
125         // This initiates the discovery process
126         if (!isConnected) bluetoothServiceDiscovery.doDeviceDiscovery();
127     }
128
129
130     /**
131      * Called when the same MIDlet is found on a remote device.
```

```
132     * It is called from BluetoothServiceDiscovery.serviceSearchCompleted().
133   *
134   * @param input Either a ServiceRecord or a StreamConnection that describes the characteristics
135     * of the Bluetooth service found
136   */
137  public void nodeFound(Object input){
138
139         // Input type is object because superclass cannot relate to ServiceRecord which is a bluetooth specific class.
140         ServiceRecord serviceRecord = (ServiceRecord)input;
141
142         // Retrives the name of the found node
143         String remoteNodeName = getRemoteNodeName(serviceRecord);
144
145         // The bluetooth address of the node "I" "as owner" found
146         String address = serviceRecord.getHostDevice().getBluetoothAddress();
147
148         // If the node is not already connected, the MIDlet is notified via the frontEnd
149         // and the user is asked whether he/she wants to connect to it or not.
150         if(getFrameworkFrontEnd().getGroup().getNode(address) == null){
151             // Alerts the FrameworkFrontEnd about the found (participating) node
152             getFrameworkFrontEnd().notifyAboutFoundNode(address,remoteNodeName);
153
154             // Saves the found node so a connection can be established later by running the connectToNodes() method.
155             foundNodes.put(address,serviceRecord);
156
157         }else{
158             // This code is run if the node has been disconnected temporarily or lacks
159           // a connection. This method (nodeFound) is then called as a result from a
160           // new serviceSearch on a given address (see BluetoothServiceDiscovery.startServiceSearch()).
161           // It re-opens a connection to a node that has been disconnected temporarily.
162           StreamConnection connection = null;
163             try{
164                 // Re-opens a connection to a (participating) node.
165               connection = (StreamConnection) Connector.open(serviceRecord.getConnectionURL(
166                                                 ServiceRecord.NOAUTHENTICATE_NOENCRYPT,false));
167             }catch(IOException ioe){
168                 // The connection could not be established
169               log.logException("BluetoothNetwork.nodeFound()", ioe, false);
170             }
171
172             getFrameworkFrontEnd().getGroup().getNode(address).setNodeConnection(connection);
173             getFrameworkFrontEnd().getGroup().getNode(address).startNodeConnection();
174         }
175     }
176
```

```
177     /**
178      *
179      * This method fetches the name of the remote node.
180      *
181      * @param input An object representing the connection to the found node.
182      * @return The name of the remote node.
183      */
184     public String getRemoteNodeName(Object input){
185         DataElement data = ((ServiceRecord)input).getAttributeValue(0x0100);
186         return (String)data.getValue();
187     }
188
189
190     /**
191      *
192      * This method establishes a connection to the chosen node.
193      * It is run from the BluetoothNetwork.sendDataPackage().
194      *
195      * @param nodeAddress The address to the node to connect to
196      *
197      */
198     public void connectToNode(String nodeAddress){
199
200       // Connects to the recipient
201         bluetoothServiceDiscovery.startServiceSearch(nodeAddress);
202
203         serviceSearchCompleted = false;
204         while(!serviceSearchCompleted){
205             // Waiting for the agent to set serviceSearchCompleted = true in
206             // the BluetoothNetwork.serviceSearchCompleted() method.
207             // This because we dont want to send the package before the search is completed
208           try{
209                 Thread.sleep(300);
210             }catch(InterruptedException ie){
211                 // Do nothing
212             }
213         }
214         // Resets the value
215         serviceSearchCompleted = false;
216
217         if (!serviceSearchFailed){
218             log.logConnection("Successfully connected to "+getFrameworkFrontEnd().getGroup().getNode(
219                                                         nodeAddress).getNodeName()+"("+nodeAddress+")");
220         }
221     }
```

```
222
223
224 /**
225  *
226  * Sets the boolean serviceSearchCompleted = true.
227  * This value will interrupt the while-loop in sendDataPackage.
228  * This because the serviceSearch must be completed before we
229  * try to send a package.
230  * The method is called from
231  * BluetoothServiceDiscovery.serviceSearchCompleted().
232  *
233  */
234 public void serviceSearchCompleted(){
235         // Resets the serviceSearchFailed in case an earlier search has failed
236   serviceSearchFailed = false;
237   serviceSearchCompleted = true;
238 }
239
240
241 /**
242  * What to do when something went wrong during servicediscovery.
243  * The method is called from
244  * BluetoothServiceDiscovery.serviceSearchCompleted().
245  *
246  */
247 public void serviceDiscoveryError(){
248   // Stops the sending of the datapackage in sendDatapackage()
249   serviceSearchFailed = true;
250   serviceSearchCompleted = true;
251 }
252
253   /**
254    *
255    * This method returns the node address.
256    *
257    * @param input String "localNode" to retreive the address of the local device.
258    * A ServiceRecord or StreamConnection object to retreive the address of a
259    * remote device.
260    *
261    * @return The node network address.
262    * @throws IOException
263    */
264   public String getNodeAddress(Object input) throws IOException{
265
266   // Checks whether the input is a String and the String equals "localnode".
```

```
267     // If so the address of the LocalDevice is returned.
268     if(input.getClass().isInstance(new String())){
269       String inputString = (String)input;
270       // Make sure that we won't get any UPPER/lower case problems
271       inputString.toLowerCase();
272       if(inputString.equals("localnode"))return LocalDevice.getLocalDevice().getBluetoothAddress();
273     }
274
275         // Input type is object because superclass cannot relate to ServiceRecord which is a bluetooth specific class.
276         // This method is valid either the input type is ServiceRecord or StreamConnection.
277         RemoteDevice remoteDevice = null;
278         try{
279             ServiceRecord serviceRecord = (ServiceRecord) input;
280             remoteDevice = serviceRecord.getHostDevice();
281         }catch(ClassCastException cce1){
282             // Could not cast the input object to ServiceRecord. Trying streamConnection instead ;-)
283             try{
284                 StreamConnection streamConnection = (StreamConnection) input;
285                 remoteDevice = RemoteDevice.getRemoteDevice(streamConnection);
286             }catch(ClassCastException cce2){
287                 //This will only happen if the input object type is wrong
288                 log.logException("BluetoothNetwork.getNodeAddress()",cce2,false);
289             }catch(IOException ioe){
290                 log.logException("BluetoothNetwork.getNodeAddress()",ioe,false);
291                 throw ioe;
292             }
293         }
294         return remoteDevice.getBluetoothAddress();
295     }
296
297     /**
298      *
299      * This method is used by the FrameworkFrontEnd to send a data package of
300      * any sort to a remote node.
301      *
302      * @param dataPackage The data package to be sent
303      * @param recipients A list containing addresses to the recipient nodes
304      *
305      */
306     public void sendDataPackage(DataPackage dataPackage, String[] recipients){
307
308       // A Vector containing the addresses to the nodes that could not be reached
309       Vector addressesToLostNodes = new Vector();
310
311       for(int i=0; i<recipients.length; i++){
```

```
312         // If the node has been removed/disconnected in the meantime
313         if (getFrameworkFrontEnd().getGroup().getNode(recipients[i])==null){
314           // do nothing
315         }else{
316
317           // Connects to the remote node if the connection never has been opened or if it has been closed
318           NodeConnection nodeConnection = getFrameworkFrontEnd().getGroup().getNode(recipients[i]).getNodeConnection();
319           if (nodeConnection!=null){
320             if (nodeConnection.getConnection()==null){
321               // Establishes a connection to the recipient
322               // This method waits until the new connection is ready (or not)
323             connectToNode(recipients[i]);
324             }else if (nodeConnection.getSendQueueSize() == 0){
325               // If the que is empty, the connection has been closed, and we need a new one
326               nodeConnection.setConnection(null);
327               // Establishes a connection to the recipient
328               // This method waits until the new connection is ready (or not)
329             connectToNode(recipients[i]);
330             }
331           }else{
332           // Establishes a connection to the recipient
333             // This method waits until the new connection is ready (or not)
334             connectToNode(recipients[i]);
335           }
336
337           // Sends the data package to the recipient
338         if (!serviceSearchFailed){
339                 getFrameworkFrontEnd().getGroup().getNode(recipients[i]).getNodeConnection().sendDataPackage(dataPackage);
340         }else{
341           // If the serviceSearch failed, the node must be removed from the group, and groups become synchronized
342           addressesToLostNodes.addElement(recipients[i]);
343         }
344         }
345     }
346       // Removes the nodes that could not be reached to remove these from the group by running a groupsync
347       for (int i=0; i<addressesToLostNodes.size();i++){
348         // Notifies only if the node is not already removed from the local group.
349         // This because a node could have been removed when sending the previous data package and this
350             // package is sent right after the first one (as in text first and then sync package)
351         if (getFrameworkFrontEnd().getGroup().getNode((String)addressesToLostNodes.elementAt(i))!=null){
352           getFrameworkFrontEnd().notifyAboutLostNode((String)addressesToLostNodes.elementAt(i));
353         }
354       }
355     }
356
```

```
357
358     /**
359      *
360      * This method returns the UUID string used as an identifier in the discovery process.
361      * The UUID string is generated based on the application ID given by the application
362      * running the framework. The UUID must be used to ensure that all nodes
363      * joining the network are running the same application.
364      *
365      * @return uuidString
366      */
367     public String getUUIDString(){
368
369         // The ASCII to Hex converter
370         ASCIIToHexConvert convert = new ASCIIToHexConvert();
371         // The String to convert to hex
372         String toConvert = "";
373         // The UUID consists of 16 hex values so the String to be converted must not exceed 16 chartacters
374         if(super.getApplicationId().length() > 16){
375           toConvert = super.getApplicationId().substring(0, 15);
376         } else {
377           toConvert = super.getApplicationId();
378         }
379         // The converted hex String
380         String convertedString = convert.convertASCIIToHex(toConvert);
381         // The length of convertedString
382         int convertedLength = convertedString.length();
383
384         // The String to return
385         String uuidString = "";
386         if(convertedLength < 32){
387           uuidString = generatedUuid.substring(0 , (32-convertedLength)) + convertedString;
388         } else if(convertedLength == 32){
389           uuidString = convertedString;
390         }
391         return uuidString;
392     }
393
394
395 }
```

D.5.2 Interface BluetoothServiceDiscoveryListener

```java
package peer2me.network.bluetooth;



/**
 * This interface has to be implemented by classes that wants to do a
 * Bluetooth service discovery using the BluetoothServiceDiscovery class,
 * and receive callbacks from this class. In this case, the class
 * BluetoothNetwork implements this interface.
 *
 * @author Torbjørn Vatn & Steinar A. Hestnes
 */
public interface BluetoothServiceDiscoveryListener {


  /**
   *
   * What to do when serviceSearch is completed
   *
   */
  public void serviceSearchCompleted();


  /**
   * What to do when something went wrong during servicediscovery
   *
   */
  public void serviceDiscoveryError();

}
```

D.5.3 Class BluetoothServiceDiscovery

```
package peer2me.network.bluetooth;

import java.util.Vector;
import javax.bluetooth.BluetoothStateException;
import javax.bluetooth.DeviceClass;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.DiscoveryListener;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.RemoteDevice;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.UUID;

import peer2me.network.Network;
import peer2me.util.Log;

/**
 * This class is responsible for doing the low level Bluetooth discovery operations.
 * The class initializes seqential device discovery, and searches for services
 * (the same MIDlet built upon the Peer2Me framework) on each of the found devices.
 *
 * @author Torbjørn Vatn & Steinar A. Hestnes
 */
public class BluetoothServiceDiscovery implements DiscoveryListener{

  // A Log instance
  Log log = Log.getInstance();

  // The current network
  private BluetoothNetwork currentNetwork;

  // A table containing the UUIDs (Universally Unique Identifier) use to perform the discovery process
  private UUID[] uuids = new UUID[1];
  // The UUID String fetched from currentNetwork
  private String uuidString;

  // Vector containing the devices found during discovery
  private Vector devicesFound = null;
  // Vector containing the services (read; running the Peer2Me framework) found on the discovered devices
  private Vector servicesFound = null;

  // An instance representing the local bluetooth device
  private LocalDevice localDevice;
  // The discovery agent of the local device
```

```
44  private DiscoveryAgent agent;
45
46  // An identifier used on mobile phone bluetooth devices
47    private int mobileDeviceClassCode = 0x200;
48    // The attributes to include in the agent.searchServices() method
49    private int[] attributes = {0x100,0x101,0x102};
50
51
52  /**
53   *
54   * Constructor.
55   * Called from BluetoothNetwork.init().
56   *
57   */
58  public BluetoothServiceDiscovery(){
59    this.currentNetwork =(BluetoothNetwork)Network.getInstance();
60    // Fetches the UUID string from currentNetwork
61    this.uuidString = currentNetwork.getUUIDString();
62  }
63
64  /**
65   *
66   * This method starts the discovery process.
67   * It is called from BluetoothNetwork.searchForNodes().
68   *
69   * @throws BluetoothStateException Error getting reference to LocalDevice
70   */
71    public void doDeviceDiscovery() throws BluetoothStateException{
72
73      uuids[0] = new UUID(uuidString, false);
74      servicesFound = new Vector();
75      devicesFound = new Vector();
76
77        try{
78            localDevice = LocalDevice.getLocalDevice();
79        }catch(BluetoothStateException bse) {
80            log.logException("BluetoothServiceDiscovery.doDeviceDiscovery()",bse, false);
81            throw bse;
82        }
83
84        //Fetches the discovery agent of the local device
85        agent = localDevice.getDiscoveryAgent();
86
87        try {
88          // The discovery agent starts the inquiry for other devices
```

```
89              agent.startInquiry(DiscoveryAgent.GIAC,this);
90          }
91          catch(BluetoothStateException bse) {
92            log.logException("BluetoothServiceDiscovery.doDeviceDiscovery()",bse, false);
93          throw bse;
94          }
95      }
96
97      /**
98       *
99       * This method is called by the javax.bluetooth.DiscoveryAgent (agent) whenever a bluetooth device is discovered
100      *
101      * @param remoteDevice The device discovered
102      * @param deviceClass The device class of the discovered device
103      *
104      */
105     public void deviceDiscovered(RemoteDevice remoteDevice,DeviceClass deviceClass){
106
107     // The device class of the discovered device
108     int deviceclass = deviceClass.getMajorDeviceClass();
109
110     if(deviceclass==mobileDeviceClassCode){
111       // Adds the discovered device to the devicesFound Vector
112       devicesFound.addElement(remoteDevice);
113     }
114     }
115
116     /**
117      *
118      * This method is called by the javax.bluetooth.DiscoveryAgent (agent) whenever one or more
119      * services (read: Peer2Me framework) are found on a remote device
120      *
121      * @param transId The transaction ID of the service search that is posting the result
122      * @param serviceRecord A list of services found during the search request
123      *
124      */
125     public void servicesDiscovered(int transId, ServiceRecord[] serviceRecord) {
126
127         // Checks whether the service already exists in servicesFound. This because the
128         // DiscoveryAgent sometimes finds the same service several times :-( STUPID!!!
129         boolean alreadyAdded = false;
130
131     for(int i=0;i<serviceRecord.length;i++){
132             if(servicesFound.size()==0){
133                 // Adds the retrived ServiceRecord to the servicesFound Vector
```

```
134                 servicesFound.addElement(serviceRecord[i]);
135                 log.logDebugInfo("BluetoothServiceDiscovery.servicesDiscovered()","Found a node with address: "+
136                         serviceRecord[i].getHostDevice().getBluetoothAddress()+" running the same application");
137             }else{
138                 for(int j=0;j<servicesFound.size();j++){
139                     // Must compare the addresses of the devices to avoid adding the same service on the same device twice or more
140                     String addressServiceFound = ((ServiceRecord) servicesFound.elementAt(j)).getHostDevice().getBluetoothAddre
141                     String addressServiceRecord = serviceRecord[i].getHostDevice().getBluetoothAddress();
142
143                     if(addressServiceFound.equals(addressServiceRecord)){
144                         alreadyAdded = true;
145                     }
146                 }
147                 if(!alreadyAdded){
148                     // Adds the retrived ServiceRecord to the servicesFound Vector
149                     servicesFound.addElement(serviceRecord[i]);
150                     log.logDebugInfo("BluetoothServiceDiscovery.servicesDiscovered()","Found a node with address: "+
151                             serviceRecord[i].getHostDevice().getBluetoothAddress()+" running the same application");
152                 }
153             }
154         }
155     }
156
157
158     /**
159      *
160      * This method is called by the javax.bluetooth.DiscoveryAgent (agent) when the search for
161      * services (read: Peer2Me framework) is completed
162      *
163      * @param transID The transaction ID of the service search that is posting the result
164      * @param respCode The response code that indicates the status of the transaction
165      *
166      */
167     public void serviceSearchCompleted(int transID, int respCode){
168
169         switch(respCode) {
170
171             case DiscoveryListener.SERVICE_SEARCH_COMPLETED:
172                 log.logDebugInfo("BluetoothServiceDiscovery.serviceSearchCompleted()","Service search completed");
173                 break;
174
175             case DiscoveryListener.SERVICE_SEARCH_DEVICE_NOT_REACHABLE:
176                 log.logDebugInfo("BluetoothServiceDiscovery.serviceSearchCompleted()","Service search device not reachable");
177                 // If a searchServices() call is made on a specific device, the devices found table will contain no devices
178                 // In this case the network must be notified about the error.
```

```
179            // In the initial doDeviceDiscovery() these errors are ignored because devices not running the framework
180                  // can interfere the discovery process.
181            if (devicesFound.size()==0) currentNetwork.serviceDiscoveryError();
182            break;
183
184         case DiscoveryListener.SERVICE_SEARCH_ERROR:
185            log.logDebugInfo("BluetoothServiceDiscovery.serviceSearchCompleted()","Service search error");
186            if (devicesFound.size()==0) currentNetwork.serviceDiscoveryError();
187         break;
188
189         case DiscoveryListener.SERVICE_SEARCH_NO_RECORDS:
190            log.logDebugInfo("BluetoothServiceDiscovery.serviceSearchCompleted()",
191                           "No bluetooth devices running the same service (application) found");
192            if (devicesFound.size()==0) currentNetwork.serviceDiscoveryError();
193            break;
194
195              case DiscoveryListener.SERVICE_SEARCH_TERMINATED:
196                 log.logDebugInfo("BluetoothServiceDiscovery.serviceSearchCompleted()","Service search terminated");
197            if (devicesFound.size()==0) currentNetwork.serviceDiscoveryError();
198         break;
199      }
200
201      // Searches further on the next device
202      if (devicesFound.size()>0){
203         try {
204            // The discovery agent searches for services on the next device stored in the devicesFound Vector
205            agent.searchServices(attributes,uuids,(RemoteDevice)devicesFound.firstElement(),this);
206            devicesFound.removeElementAt(0);
207         } catch (BluetoothStateException bse) {
208            log.logException("BluetoothServiceDiscovery.serviceSearchCompleted", bse, true);
209            currentNetwork.serviceDiscoveryError();
210         }
211      }
212
213      else{
214         if (servicesFound.size()==0){
215            log.logDebugInfo("BluetoothServiceDiscovery.serviceSearchCompleted()","No services found");
216         }else{
217                  log.logDebugInfo("BluetoothServiceDiscovery.serviceSearchCompleted()",
218                                 "Found the desired service running on one or more nodes");
219            // For each element in servicesFound the serviceFound method is called on currentNetwork
220                  for(int i=0;i<servicesFound.size();i++){
221                     currentNetwork.nodeFound((ServiceRecord)servicesFound.elementAt(i));
222            }
223            currentNetwork.serviceSearchCompleted();
```

```
224         }
225       }
226     }
227
228
229     /**
230      *
231      * This method is called by the javax.bluetooth.DiscoveryAgent (agent) when the discovery process is completed
232      *
233      * @param discType The type of request that was completed; either
234      * INQUIRY_COMPLETED, INQUIRY_TERMINATED, or INQUIRY_ERROR
235      *
236      */
237     public void inquiryCompleted(int discType) {
238
239       switch (discType) {
240
241     case DiscoveryListener.INQUIRY_COMPLETED:
242       if(devicesFound.size()==0){
243         log.logDebugInfo("BluetoothServiceDiscovery.inquiryCompleted()","No devices found");
244                 // Send a message to the midlet
245         currentNetwork.serviceSearchCompleted();
246       }else{
247         try {
248                     log.logDebugInfo("BluetoothServiceDiscovery.inquiryCompleted()","Found one or more devices");
249           // The discovery agent searches for services on the first device stored in the devicesFound Vector
250           agent.searchServices(attributes,uuids,(RemoteDevice)devicesFound.firstElement(),this);
251           devicesFound.removeElementAt(0);
252         } catch (BluetoothStateException bse) {
253             log.logException("BluetoothServiceDiscovery.serviceSearchCompleted", bse, true);
254         }
255       }
256     log.logDebugInfo("BluetoothServiceDiscovery.inquiryCompleted()","Device inquiry completed");
257     break;
258
259     case DiscoveryListener.INQUIRY_ERROR:
260
261     log.logDebugInfo("BluetoothServiceDiscovery.inquiryCompleted()","Device inquiry error");
262     break;
263
264     case DiscoveryListener.INQUIRY_TERMINATED:
265
266     log.logDebugInfo("BluetoothServiceDiscovery.inquiryCompleted()","Device inquiry terminated");
267     break;
268     }
```

```
269     }
270
271
272     /**
273      *
274      * This method is used to re-establish a connection to a device when we have the address.
275      *
276      * @param address The address to the device
277      */
278     public void startServiceSearch(String address){
279         // Connects to a remote device with the given address
280         RemoteDeviceInstance remoteDevice = new RemoteDeviceInstance(address);
281         try{
282             localDevice = LocalDevice.getLocalDevice();
283             agent = localDevice.getDiscoveryAgent();
284             uuids[0] = new UUID(uuidString, false);
285             servicesFound = new Vector();
286             devicesFound = new Vector();
287             agent.searchServices(attributes,uuids,remoteDevice,this);
288         }catch(BluetoothStateException bse) {
289             log.logException("BluetoothServiceDiscovery.startServiceSearch()",bse, false );
290             // throw bse;
291         }
292     }
293
294
295     /**
296      *
297      * This private class creates a RemoteDevice based on the address.
298      * It is used during the re-establishment of a connection to a device in
299      * the startServiceSearch() method in this class.
300      *
301      * @author Torbjørn Vatn & Steinar A. Hestnes
302      */
303     private class RemoteDeviceInstance extends RemoteDevice{
304
305         /**
306          *
307          * Constructor. Forwarding the address to the superclass.
308          *
309          * @param address The address of the device
310          */
311         public RemoteDeviceInstance(String address){
312             super(address);
313         }
```

```
    }


}
```

Bibliography

[1] Jim Arlow and Ila Neustadt. *UML and the Unified Process*. Addison-Wesley, 2002.

[2] Roy L. Ashok and Dharma P. Agrawal. Next-generation wearable networks. *IEEE Computing Practices Magazine*, pages 31–39, nov 2003.

[3] Stefano Basagni, Marco Conti, Silvia Giordano, and Ivan Stojmenovic. *Mobile ad hoc Networking*. IEEE, 2004.

[4] V.R. Basili. The Experimental Paradigm in Software Engineering. *Experimental Software Engineering Issues: Critical Assessment and Future Directions*, pages 3–12, 1992.

[5] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice, second edition*. Addison Wesley, 2003.

[6] Peter H. Carstensen and Kjeld Schmidt. Computer supported cooperative work: New challenges to systems design. *Handbook of Human Factors*, 2002.

[7] JXTA Community. JXTA.org. `http://www.jxta.org`. 03.03.2006.

[8] BEDD Corporation. BEDD Corporation. `http://www.bedd.com/`. 06.03.2006.

[9] Zürich Ergon Informatik AG. Ergon. `http://ergon.ch/`. 05.03.2006.

[10] ETH. Jadabs. `http://jadabs.berlios.de/`. 03.03.2006.

[11] George H. Forman and John Zahorjan. The challenges of mobile computing. *Computer*, 27(4):38–47, 1994.

[12] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[13] JSR259 Expert Group. JSR259 Ad Hoc Networking API Early Draft Review. review 1, BenQ Mobile GmbH, 2006.

[14] Jonathan Grudin. Cscw. *Commun. ACM*, 34(12):30–34, 1991.

[15] Jonathan Grudin. Computer-supported cooperative work: History and focus. *Computer*, 27(5):19–26, 1994.

[16] Jaap C. Haarsten. The Bluetooth Radio System. *IEEE Personal Communications*, 2000.

[17] Sumi Helal. Pervasive java. *IEEE Pervasive Computing*, 1(1):82–85, 2002.

[18] Little Spring Design Inc. Mobile Style Guides - Multiple-Device Platforms. `http://www.littlespringsdesign.com/design/styleguides/sampleguide/sampleguide2/`. 12.12.2005.

[19] Christina L. James and Kelly M. Reischel. Text input for mobile devices: comparing model prediction to actual performance. In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 365–371, New York, NY, USA, 2001. ACM Press.

[20] R. Johansen. Current user approaches to groupware. *In R. Johansen (ed.), Groupware : Computer support for business teams.*, pages 12–44, 1988.

[21] Ralph E. Johnson and Brian Foote. Designing Reuseable Classes. review 1, Department of Computer Science, University of Illinois, 1991.

[22] Per Kroll and Phillippe Kruchten. *The Rational Unified Process Made Easy.* Addison-Wesley, 2003.

[23] Yi Lu and Serge Vaudenay. Faster correlation attack on bluetooth keystream generator e0. In *Advances in Cryptology ? CRYPTO 2004: 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004.*, page 407. Springer Berlin / Heidelberg, 2004.

[24] Nico Maibaum and Thomas Mundt. Jxta: A technology facilitating mobile peer-to-peer networks. In *MOBIWAC '02: Proceedings of the International Workshop on Mobility and Wireless Access*, page 7, Washington, DC, USA, 2002. IEEE Computer Society.

[25] Sun Microsystems. J2ME Building Blocks for Mobile Devices - White Paper on KVM and the Connected, Limited Device Configuration (CLDC). `http://java.sun.com/products/cldc/wp/KVMwp.pdf`. 02.12.2005.

[26] SUN Microsystems. Java 2 Platform, Micro Edition (J2ME); JSR 68 Overview. `http://java.sun.com/j2me/overview.html`. 15.09.2005.

[27] Claes Wohlin Mikael Svahnberg. An investigation of a method for identifying a software architecture candidate with respect to quality attributes. *Empirical Software Engineering*, 10:149–181, apr 2005.

[28] Michael Miller. *Discovering Bluetooth, Learn What You Can Do with Bluetooth Today - and What You'll Be Able to Do Tomorrow* . SYBEX, 2001.

[29] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. Peer-to-peer computing. Technical report, HP Laboratories Palo Alto, 2002.

[30] Michael Sars Norum and Carl-Henrik Wolf Lund. A Framework for Mobile Collaborative Applications on Mobile Phones. Depth study, IDI, NTNU, 2004. This is the depth study that was the base for the master thesis.

[31] Michael Sars Norum and Carl-Henrik Wolf Lund. The Peer2Me Framework; A Framework for Mobile Collaboration on Mobile Phones. Master's thesis, IDI, NTNU, 2005.

[32] Kris Read and Frank Maurer. Developing mobile wireless applications. *IEEE Internet Computing Magazine*, pages 81–86, jan-feb 2003.

[33] Yaniv Shaked and Avishai Wool. Cracking the bluetooth pin. In *MobiSys '05: Proceedings of the 3rd international conference on Mobile systems, applications, and services*, pages 39–50, New York, NY, USA, 2005. ACM Press.

[34] Forrest Shull, Jeffrey Carver, and Guilherme H. Travassos. An empirical methodology for introducing software processes. In *ESEC/FSE-9: Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 288–296, New York, NY, USA, 2001. ACM Press.

[35] Bluetooth Special Interest Group (SIG). Bluetooth 2.0 Facts and Features. `https://www.bluetooth.org/info/edr/edr_facts.php`. 15.12.2005.

[36] Bluetooth Special Interest Group (SIG). Bluetooth SIG promoter members. `https://www.bluetooth.org/foundry/sitecontent/document/member_directory`. 15.12.2005.

[37] Statistics Norway SSB. Telecommunication services. 31 December 2005. `http://www.ssb.no/english/yearbook/tab/tab-439.html`. 08.02.2006.

[38] Anders Rene Sveen and Lars Kirkhus. MOWAHS - Mobile Collaboration Framework, 2004.

[39] Torbjørn Vatn and Steinar Hestnes. The Peer2Me Framework; An Analysis of the Peer2Me Framework. Depth study, IDI, NTNU, 2005. This is our own depth study.

[40] E. Vergetis, R. Guerin, S. Sarkar, and J. Rank. Can Bluetooth succeed as a large-scale ad hoc networking technology? *Selected Areas in Communications, IEEE Journal on*, 23:644– 656, 3 2005.

[41] G. Caldiera V.R. Basili and H.D. Rombach. *Goal Question Metrics Paradigm*, volume 1. Wiley, 1994.

[42] Laurie A. Williams and Robert R. Kessler. All i really need to know about pair programming i learned in kindergarten. *Commun. ACM*, 43(5):108–114, 2000.

[43] Claes Wohlin, Per Runeson, Martin Host, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslen. *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.