Evaluation of Peer2Me

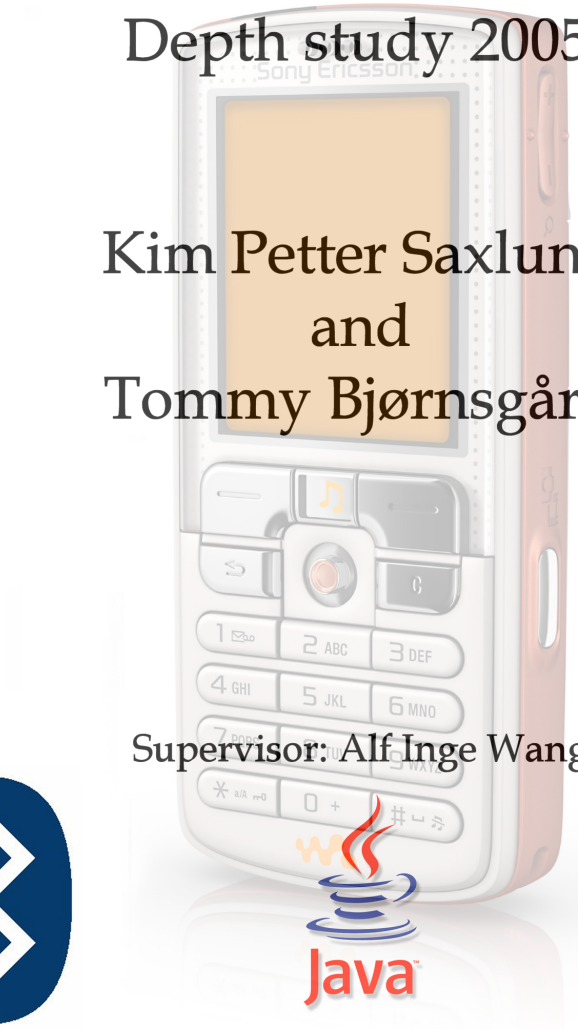
TDT4735 Software Engineering

Depth study 2005

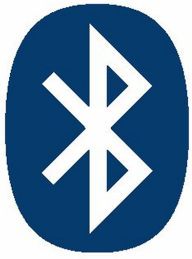
Kim Petter Saxlund

and

Tommy Bjørnsgård



Supervisor: Alf Inge Wang



Abstract

This project evaluates the Peer2Me framework, a framework for collaborative applications on mobile phones that utilize Personal Area Networks(PANs). Peer2Me is the work of Carl-Henrik Wolf Lund and Michael Sars Norum carried out in their master thesis in 2005.

This paper describes central, theoretical concepts related to Peer-to-Peer(P2P) computing, Mobile Ad Hoc NETworks (MANETs) and Computer Supported Cooperative Work (CSCW) domains, focusing on presence-collaboration.
Evaluation methods used in this project are presented along with four developer research questions and seven end-user questions.

In order to find answers to the research questions, the report describes two applications that have been developed using the Peer2Me framework. The applications are called PeerQuiz and PeerShare. These applications have been tested both by the developers and by a test-group. During the process of this project, weaknesses and fields of improvements in the Peer2Me framework have been discovered. These problems are described in detail in the evaluation part of this report.

This report also contains a prestudy that describes essential cooperation theory and technology, related technology, details about the Peer2Me framework and related projects.

We argue how PAN-technology enables a broad range of collaborative applications supporting both collocated and spontaneous interaction.
Description and evaluation of relevant technologies and projects related to Peer2Me is also described.

The source code and deployable packages of PeerQuiz and PeerShare can be found on the attached CD found in the back of this report.

Preface

This project is the result of the work carried out by Kim Saxlund and Tommy Bjørnsgård from August to December 2005. The project is a part of the course TDT4735 Software Engineering, Depth Study. The course is a part of the fifth year of the Master in Technology degree in Computer Science at the Norwegian University of Science and Technology. The project descriptions was assigned by the Department of Computer and Information Science.

Acknowledgements

We would like to thank Alf Inge Wang for his invaluable guidance during the writing of this report. We would also like to thank Michael Sars Norum and Carl-Henrik Wolf Lund for valuable information regarding the Peer2Me framework.

We would also like to thank the participants of the end-user testing and for letting us borrow their mobile phones : Torbjørn Vatn, Steinar Anders Hestnes, Jørgen Rygh and Even Andrè Fiskvik.

Trondheim December 19th, 2005

Kim Saxlund

Tommy Bjørnsgård

Contents

List of Tables **ix**

List of Figures **x**

I Introduction **1**

1 Motivation **3**
1.1 Problem definition 4

2 Project context **5**

II Research questions and methods **7**

3 Research questions **9**
3.1 Research questions 9
3.2 Evaluation plan 11
3.2.1 Research 12
3.2.2 Design 12
3.2.3 Implementation 12
3.2.4 Testing 12
3.2.5 Evaluation 12
3.3 Test-application evaluation method 13

4 Development method and software **15**
4.1 Development methods 15
4.2 Development tools 15
4.2.1 Eclipse with plugins 16
4.2.2 Java Wireless Toolkit (J2ME) 16
4.2.3 MiKTeX 16
4.2.4 Concurrent Versioning System 16

5 Requirements elicitation and process **17**
5.1 Scenario 17
5.2 Goals 17
5.3 Functional requirements 17
5.4 Use case diagram 18
5.5 Quality requirements 18

III Prestudy **19**

6 Essential Cooperation Theory and Technology **21**
6.1 Peer-to-peer 21
6.2 Mobile P2P networks 23

6.3 Mobile Ad Hoc Networks (MANET) . . . 23
6.4 Communication and Collaboration . . . 26
6.4.1 Groupware and Computer Supported Cooperative Work (CSCW) . . . 26

7 Peer2Me **29**
7.1 Introduction . . . 29
7.2 Design . . . 29
7.2.1 Domain Concepts . . . 29

8 State of the art **33**
8.1 Bluecove . . . 33
8.2 Bluetooth Ad-hoc Networking for Inter-Vehicle Communication . . . 33
8.3 Umbrella.net . . . 34
8.4 BEDD . . . 35
8.5 Other projects . . . 35

9 Technology **37**
9.1 Mobile phones . . . 37
9.2 Java 2 Micro Edition . . . 38
9.2.1 J2ME architecture . . . 38
9.2.2 Optional packages . . . 39
9.3 Wireless network technologies . . . 39
9.3.1 Bluetooth . . . 40
9.3.2 Zigbee . . . 40
9.3.3 Radio Frequency IDentification . . . 41
9.3.4 Wireless Local Area Network . . . 41
9.3.5 Comparison . . . 42
9.4 XML . . . 44
9.4.1 Strengths and weaknesses . . . 44
9.4.2 Syntax . . . 45
9.4.3 Validation . . . 45
9.4.4 XML parsing . . . 46

IV Test application 1 - PeerQuiz **49**

10 Introduction **51**

11 Requirements **53**
11.1 Scenario . . . 53
11.2 Goals . . . 53
11.3 Functional requirements . . . 54
11.4 Usecase description . . . 55
11.5 Quality requirements . . . 56
11.5.1 Usability . . . 56
11.5.2 Testability . . . 56
11.5.3 Modifiability . . . 56
11.5.4 Availability . . . 57

12 Dependencies **59**
12.1 Packages . . . 59
12.2 Supported mobile phones . . . 60

13 Design **61**
13.1 The PeerQuiz Package . . . 61
13.2 The Game Package . . . 62

13.3 The Gui Package . . . 62
13.4 The Question Package . . . 63
13.5 The Util Package . . . 63
13.6 Class Diagrams . . . 64
13.7 Program flow . . . 71

14 Implementation **73**
14.1 Xml parsing and Questions.xml . . . 73
14.1.1 kXml Pull Parsing . . . 74
14.1.2 Put to use . . . 74
14.2 Availability tactics and ping-functionality . . . 74
14.2.1 Ping/echo . . . 75
14.2.2 QuizTimer . . . 75
14.3 GUI - Graphical User Interface . . . 75
14.3.1 Form . . . 75
14.3.2 The PeerQuiz progressbar, BarItem . . . 76
14.4 Communication . . . 76
14.4.1 Initialization . . . 76
14.4.2 Master vs. Slave . . . 77
14.4.3 Sending messages . . . 78
14.5 Code statistics . . . 79

15 Testing **81**
15.1 Test of functional requirements . . . 81
15.1.1 Comments on the implementation of the requirements . . . 82
15.2 Test of quality requirements . . . 83

V Test application 2 - PeerShare **85**

16 Introduction **87**

17 Requirements **89**
17.1 Scenario . . . 89
17.2 Goals . . . 89
17.3 Functional requirements . . . 90
17.4 Usecase description . . . 90
17.5 Quality requirements . . . 90
17.5.1 Usability . . . 91
17.5.2 Testability . . . 91
17.5.3 Modifiability . . . 91
17.5.4 Availability . . . 91

18 Dependencies **93**
18.1 Packages . . . 93
18.2 Supported mobile phones . . . 94

19 Design **95**
19.1 The peershare package . . . 95
19.2 The peershare.gui package . . . 96
19.3 The peershare.util package . . . 96
19.4 Class Diagrams . . . 98
19.5 Program flow . . . 101

20 Implementation **103**
20.1 FileConnection APIs . . . 103

20.2 Xml parsing . . . 104
20.3 GUI - Graphical user interface . . . 105
20.4 Communication . . . 105
20.4.1 Initialization . . . 105
20.4.2 Setting up the filesharing network . . . 106
20.4.3 Downloading files from others . . . 107
20.4.4 The truth about Peer2Me communication . . . 107
20.5 Storing application data . . . 108
20.6 Code statistics . . . 108

21 Testing **109**
21.1 Test of functional requirements . . . 109
21.1.1 Comments on the implementation of the requirements . . . 109
21.2 Test of quality requirements . . . 110

VI Evaluation 113

22 Answers to research questions **115**
22.1 Developer questions . . . 115
22.2 End-user questions . . . 120
22.2.1 Answers to PeerQuiz . . . 120
22.2.2 Answers to PeerShare . . . 122
22.3 Summary . . . 123

23 Summary of the project **125**
23.1 Conclusion . . . 125
23.2 Further work . . . 125
23.2.1 Short-term goals . . . 125
23.2.2 Long-term goals . . . 126

Bibliography **127**

VII Appendix 131

A Snapshots of PeerQuiz **133**

B Snapshots of PeerShare **141**

C Questionnaire for Peer2Me developer testing **149**

D Contents of CD-Rom **153**

List of Tables

3.1 Format of buglist 10
3.2 Format of improvement list 10
3.3 Format of new functionality list 11

5.1 Description of quality attribute scenario 18

6.1 Comparison of pure and hybrid model 22
6.2 Groupware Time Space Matrix 27

7.1 Framework statistics 32

9.1 Display resolutions of mobile phone with J2ME support 38
9.2 The different WLAN standards 42
9.3 Strength and weaknesses of XML 45

11.1 Functional requirements for PeerQuiz 54

12.1 Supported mobile phones 60

14.1 PeerQuiz code statistics 79

15.1 Fulfillment of functional requirements for PeerQuiz 82

17.1 Functional requirements for PeerShare 90

18.1 Supported mobile phones 94

20.1 PeerShare code statistics 108

21.1 Fulfillment of functional requirements for PeerShare 110

List of Figures

6.1 Taxonomy of computer systems . . . 22
6.2 Ad hoc networks taxonomy . . . 24
6.3 A singlehop ad hoc network . . . 24
6.4 A multihop ad hoc network . . . 25
6.5 Scatternet comprising three piconents . . . 26
6.6 Digital spheres intersecting . . . 27

7.1 Taxonomy of computer systems . . . 30
7.2 Taxonomy of computer systems . . . 31

8.1 The GUI of the Umbrella software . . . 34

9.1 Taxonomy of computer systems . . . 39
9.2 Speed comparison . . . 43
9.3 Range comparison . . . 43
9.4 Power consumption comparison . . . 44

11.1 Use case model for PeerQuiz . . . 55

13.1 Package diagram for PeerQuiz . . . 61
13.2 The peerquiz package . . . 64
13.3 The game package . . . 65
13.4 The game package part 2 . . . 66
13.5 The gui package part 1 . . . 67
13.6 The gui package part 2 . . . 68
13.7 The question package . . . 69
13.8 The util package . . . 70
13.9 PeerQuiz flowchart . . . 71
13.10 Peer Quiz UML activity diagram . . . 72

14.1 BarItem graphics . . . 76

17.1 Use case model for PeerShare . . . 90
17.2 File transfer in PeerShare . . . 92

19.1 Package diagram for PeerShare . . . 95
19.2 The peershare package . . . 98
19.3 The gui package . . . 99
19.4 The util package . . . 100
19.5 Activity diagram of PeerShare . . . 101

20.1 Taxonomy of computer systems . . . 106
20.2 Taxonomy of computer systems . . . 107
20.3 Taxonomy of computer systems . . . 107

22.1 Mobile ad hoc collaboration in a natural environment . . . 121

A.1 Splash screen . . . 134
A.2 Main menu . . . 134
A.3 Settings menu . . . 135
A.4 Choosing quiztype . . . 135
A.5 Custom quiz . . . 136
A.6 Entering nickname . . . 136
A.7 Entering nickname . . . 137
A.8 Request to allow Bluetooth connection . . . 137
A.9 Searching for master . . . 138
A.10 Searching for slaves . . . 138
A.11 Answering questions . . . 139
A.12 Submitting answers . . . 139
A.13 Winner declared . . . 140

B.1 Main menu . . . 142
B.2 Settings menu . . . 142
B.3 Permission to access file system . . . 143
B.4 Selecting download and upload folders . . . 143
B.5 Selecting download folder . . . 144
B.6 Download folder is selected . . . 144
B.7 Download & Upload folders selected . . . 145
B.8 Entering nickname . . . 145
B.9 Request to allow Bluetooth connection . . . 146
B.10 A master's perspective - Nodes found . . . 146
B.11 A slave's perspective - Nodes found . . . 147
B.12 Browsing the connected peer's files . . . 147

Listings

9.1 Example of an xmlfile . . . 45
9.2 Example of an DTD file . . . 46
14.1 Xml-format, from `Questions.xml` . . . 73
14.2 DTD for `Questions.xml` . . . 74
14.3 Setting new screen codeexample . . . 75
14.4 Initializing the framework . . . 76
14.5 Setting listeners and handlers . . . 77
14.6 Registering components . . . 77
14.7 Searching for slavenodes . . . 77
14.8 Slavenode is discovered by masternode . . . 77
20.1 Imports in the FileHandler class, from `FileHandler.java` . . . 103
20.2 Getting content of directory, from `DirectoryBrowserScreen.java` . . . 103
20.3 Connecting to filesystem and retrieving contents, from `FileHandler.java` . . . 104
20.4 Calling the recursive method, from `FileHandler.java` . . . 104
20.5 DTD for example.xml, from `rules.dtd` . . . 104
20.6 Sample xml file, from `example.xml` . . . 105
20.7 Slave - Initializing components, from `PeerShare.java` . . . 105
20.8 Master - Initializing components and start search, from `PeerShare.java` . . . 105
20.9 Master - Starting search, from `PeerShare.java` . . . 106
20.10 Slave - joins group, from `PeerShare.java` . . . 106

Part I

Introduction

CHAPTER 1

Motivation

Numbers from SSB, Statistics Norway [28] shows that the last five years in the telecom sector, sales have never been higher, nor have the penetration of the market been greater. As the number of mobile phones are increasing, they are also getting more and more advanced. Todays mobile phones are able to perform tasks that were impossible two years ago, and a big challenge is to take advantage of the new technology. With todays technology focusing on connectivity and recreation, mobile ad-hoc communication and collaboration is a growing domain.
Creating software solutions for this domain can be difficult due to the multitude of different mobile phones on the market. With commercial software, one must be sure that whoever is using the application, uses it sucessfully. There are several processes which can aid the developer in creating good applications, whereas standarization would be the most useful. Standards are good for shortering the time-to-market as well as increasing the target area for an application. There are several standards and solutions when it comes to software on a mobile phone. Today, Java is supported by most modern mobile phones and Java 2 Platform, Micro Edition (J2ME), is proving to be de facto standard. Windows Mobile - based smartphones are also increasing in number. These mobile phones support applications developed on the .NET Compact Framework and some even support J2ME applications. With J2ME, software developers can create useful and sensible applications, promoting informal ad-hoc collaboration. Kraut, Fish, Root and Chalfonte defines informal collaboration:

> "Informal collaboration, opportunistic or spontaneous, lacks predefined agenda and schedule, is short, and involves random participants." [31]

Peer2Me is a framework which lets the developer create applications based on the principle of mobile ad-hoc communication. Framework is defined as:

> "A set of classes that embodies an abstract design for solutions to a family of related problems." [30]

Peer2Me abstracts the network layer, making the communication semi-transparent to the developers. It will be easier to create powerful applications that communicate wirelessly.
The Peer2Me framework is not ready to be released, and still needs a final touch. Our motivation is to improve the framework, discover faults and weaknesses, and take it one step closer to release. This leads to our problem definition.

1.1 Problem definition

A goal of the Peer2Me project is to distribute the framework freely over the Internet. The problem is that the framework has not been properly tested yet. Our objective is therefore to make a thorough evaluation of the Peer2Me framework. The evaluation will find out how useful Peer2Me is for developing applications for mobile phones. During this process, we expect to find issues in Peer2Me that need to be resolved. These issues may be bugs in the framework or lack of wanted functionality. The findings will most likely give proper incentive to further development with the Peer2Me framework.

CHAPTER 2

Project context

This project is a part of a research project called MOWAHS - Mobile Work Across Heterogeneous Systems. The project started in early 2001. Responsible for the project is professor Reidar Conradi and professor Mads Nygård, which both work at NTNU. The partners of the MOWAHS project are the software engineering and database research groups at IDI, NTNU. The goals of MOWAHS are threefold:

G1) Helping to understand and to continuously assess and improve workprocesses in virtual organizations.
G2) Providing a flexible, common work environment to execute and share real workprocesses 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.

This project evaluates a framework created in 2005 at NTNU, which satisfies G2. The results are distributed on the Peer2Me website[1] which then fulfills G3. There is also another group of students at NTNU who are working with the framework at the same time as us. Both groups have the same goal: evaluate the framework, but with different approaches. This project evaluates the framework by developing applications that incorporates the framework, while the other group's focus is on performance, persistence and fault tolerance. Both groups exchange knowledge and ideas with each other during the project period regarding fields of improvement and problems experienced. These encounters are informal and spontaneous since we are located in the same building. We are also communicating through a instant messaging program.

[1] http://www.peer2me.org

Part II

Research questions and methods

CHAPTER 3

Research questions

In this chapter we will identify the research questions we seek to answer in this project, and how we will answer these questions.

3.1 Research questions

In every development project, an evaluation process will often imply major challenges including choosing proper evaluation methods. An evaluation of a project should be thorough, but still easy to carry out. The evaluation should determine weaknesses and strengths in a project, and will often be used to improve or change the software. In this project, we will evaluate the Peer2Me-framework by creating applications using the framework. This will give useful knowledge about the framework, and how it can be used as building-blocks in diverse applications. This is a more practical approach to an evaluation phase and has its' advantages and disadvantages compared to more standardized evaluation methods like Scenario-based software architecture evaluation methods (SAAM) [19] being a method to evaluate architecture. While our approach best reflects how the framework can be used in real-life, a SAAM evaluation method would use different scenarios to express quality aspects as modifiability, flexibility etc. in a more isolated environment.
Since the Peer2Me framework is based on a new and immature technology, it is interesting to see if this has inflicted the performance and/or functionality of the framework. The research questions we want to answer are:

Question 1: Can a developer with some experience in Java, adopt the Peer2Me framework as a utility for developing applications for mobile phones?

Answer requirements: Sars and Norum [22] performed a workshop with a couple of students. None of the students had any experience with Java 2 Micro Edition, but managed to complete the workshop nevertheless. They also filled out a questionnaire at the end of the workshop. We will also fill out a slightly modified questionnaire and compare the results with the findings in [22]. We expect to find differences in the two findings. In the workshop, Sars and Norum were present to answer questions and help the participants. We have the ability to send e-mails to them, but it is more or less expected that the documentation in [22] is sufficient enough. Also, our applications will be much more extensive than the one in the workshop.

Question 2: Which bugs exists in the framework and what kind of impact do they have for development?

Answer requirements: The bugs should be noted as they are found during development. The bugs must be categorized into two groups: previously documented and undocumented. Each buglist must follow the format illustrated by Table 3.1.

Description of bug	A short description of the bug.
Found in class	Include the package as well as the class name the bug was found in
Affected classes	A list of any other classes that must be changed in order to correct the bug
Priority	Should indicate how important it is to fix the bug. The values are: low, medium, high
Est. time to correct	Should contain a rough estimate of time to correct based on reasonable arguments. Measured in hours.

Table 3.1: *Format of buglist*

If the bugs found actually is in the J2ME platform and not the Peer2Me, they should also be included in the answer.

Question 3: How can the framework be improved?

Answer requirements: This answer should list the bugs found in question 2 worth fixing. If there are components in the framework that seems to need improvement, a careful consideration should be made before recommending improvements. Elements that are ready for modification should be represented like Table 3.2.

Element	What element in the framework needs to be changed? Examples: messaging system, network type etc.
Classes affected	Include the packages as well as the class names that are affected
Est. time to change	Should contain a rough estimate of time to change based on reasonable arguments. Measured in hours.
Projected positive outcome	List of positive effects
Projected negative outcome	List of negative effects

Table 3.2: *Format of improvement list*

Question 4: How can we add functionality and value to the framework?

Answer requirements: This answer should list new functionality that could be implemented. The answer must also evaluate the suggested further work discussed in Chapter 21 in [22].
New functionality should be represented like Table 3.3.

Element	What element that should be implemented?
Est. time to develop	Should contain a rough estimate of time to develop based on reasonable arguments. Measured in hours.
Projected positive outcome	List of positive effects
Projected negative outcome	List of negative effects

Table 3.3: *Format of new functionality list*

To answer these questions we have to develop applications using the framework, and take notice of problems and/or bugs during the process to gather as much information as possible.

Since we are going to develop applications, we would also like to perform a usertest in order to evaluate the applications themselves. From our test-group we hope to find answers to these questions:

1. What kind of functionality does the application lack?
2. What problems did you experience?
3. What are the positive sides of the application?
4. What are the negative sides of the application?
5. What do you think is the MOST important factor for this kind of application; Usability, Stability, Performance, Functionality or Entertainment value?
6. What do you think of the usability of the application (1=poor, 10=fantastic) and why?
7. Do you see yourself using such applications often (daily, monthly, never) and why?

The answers found by these questions will be the foundation for the evaluation phase of this project. The evaluation will be the starting point in the master thesis which will be written spring 2006.

3.2 Evaluation plan

This section will describe how the evaluation process will be rendered, and which criteria that will be in focus.
There are many research methods that can be applied in software-evaluation. Basili describes three different approaches in [4]:

The engineering method: By using the engineering experimental method, engineers build and test a system according to a hypothesis. Based upon the result of the test, they improve the solution until it requires no further improvement. This method is typically used to find better methods for structuring large system.

The empirical method: A statistical method is proposed as a means to validate a given hypothesis. Data is collected to verify or falsify the hypothesis. This method is typically used when comparing a new technology against an old technology.

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

The best suited method for this project is *the engineering method*, because it is defined as the best approach when trying to improve existing software. It will also give a more practical approach and is typically used for structuring large systems.

3.2.1 Research

Before answering the research questions by developing the applications, background knowledge about the Peer2Me framework is crucial in order to plan and design sensible applications. This is done by reading the Peer2Me documentation found in the master thesis by Sars and Norum[22], and getting an overview of the available technologies.
Performing a thorough research will give good insight in the components that makes up the framework, and how they relate to each other. This will be useful when designing the test-applications.

3.2.2 Design

To evaluate the framework in the best possible way, it is important to design the test-applications to utilize as much of the framework as possible. Doing this will give a more complete overview of the framework's performance and stability. To achieve this, **two** test-applications will be developed, and a set of requirements will be elicited for each.

3.2.3 Implementation

Each of the applications will be developed independently of each other using the same Java runtime version and test-equipment. This is to ensure that the Peer2Me framework does not behave differently in one of the applications.
During the implementation all problems and issues will be written down and used to answer the research questions described in Section 3.1.

3.2.4 Testing

The test-phase will consist of two main parts:

Internal testing: Here we will test the applications ourselves to see if the requirements are met. If the requirements are not met, this will be discussed in the evaluation.

Test group: A test group which will consist of 3-6 persons should test the applications and gives us feedback through a questionnaire.

The applications will also be tested for bugs during the implementation.

3.2.5 Evaluation

The evaluation of the project will be based on several factors:

Answers to research questions: The answers to the research questions will be presented and discussed .

Requirements: Which requirements was *not* met by the application and why not.

Improvements: Which improvements can be made to the framework? This involves finding bugs in the framework.

Expansion: Did we find any lack of functionality? Here we will present new functionality that the framework will benifit from implementing.

Answers to user questionnaire: We will discuss and present how the test-group answered the questionnaire.

3.3 Test-application evaluation method

The test-applications will be evaluated by a handful of test-subjects which will answer the 7 questions listed in chapter 3.1. The first four questions requires the participants to answer freely. In question five, five alternatives are given: *Usability, Stability, Performance, Functionality* and *Entertainment value*. Question six, requires that the participant give an appropriate score (1=poor, 10=excellent) along with reasoning for the given score. In the last question, the participant will be given five alternatives: *daily, weekly, monthly, yearly* and *never*.

CHAPTER 4

Development method and software

In order to create software, methods and tools are needed. Using the proper methods and tools, time-to-market can be substantially reduced, with the focus on the quality of the product.
There are many different methods and tools to choose from, and those used in this project are described in Section 4.1 and Section 4.2.

4.1 Development methods

There are many development methods available, and two of the most common ones regarding software development are *the waterfall development method* and *XP*, *eXtreme Programming*. Both offer different approaches, and can often be used in the same situation. XP is based on values of simplicity, communication and feedback, with a focus on frequent testing and user review. The requirements are often based on user stories, and therefore less traceable.
The method most commonly used in software development project, is the waterfall method. This is an incremental development process going through several steps in a structured way. When using waterfall, it is common to divide the process into four main step:

- Elicit requirements
- Create requirements
- Design code
- Implement

We chose to use the waterfall method, since this is best suited in a team consisting of two persons, while XP works best in larger teams. The waterfall method is also providing a more structured way of creating an application.

4.2 Development tools

This section will describe the software tools used in this project, and all the software are either freeware or shareware.

4.2.1 Eclipse with plugins

Eclipse[1] is a an open source development platform with lots of available plugins for different purposes, e.g. making UML-diagrams or designing graphical user interfaces.
We had to install several plugins in order to cover our needs:

Texlipse - We wanted to write this documentation in LaTeX and used the plugin Texlipse[2], that allowed us to do this in Eclipse.

EclipseMe - It is not possible to create J2ME MIDlets in Eclipse without EclipseMe[3]. The plugin connects to a wireless toolkit that is installed on the system. This means that the emulator can be run within Eclipse.

Metrics - In order to find good code statistics we used the plugin Metrics[4].

Modelistic JME - The Modelistic JME plugin [5] automatically creates class diagrams with a wide range of different views. It is possible to show only public methods, highlight special items such as abstract classes, suppress private methods, create dependencies etc. There are plenty more options available which much be experienced to see the advantage of using it.

4.2.2 Java Wireless Toolkit (J2ME)

The EclipseMe plugin requires that a wireless toolkit is installed. For this purpose, we installed the Sun Java Wireless Toolkit 2.2 [6].
This toolkit offers a complete solution for developing JAVA MIDlet applications on a computer. It includes the libraries, emulation environments, documentation, and examples of how J2ME is utilized.

4.2.3 MiKTeX

To write in LaTeX we needed an implementation of TeX and we chose MiKTeX[7] for this purpose.
MiKTeX offers a complete set of utilities, macro packages and fonts, and can easily be used with development environments such as Eclipse.
The idea behind LaTeX is that author should be able to concentrate on *writing* within the logical structure of their document, rather than spending their time on the details of *formatting*.

4.2.4 Concurrent Versioning System

Concurrent Versioning System, CVS, is a software package thatkeeps track of different versions of files, and can handle both text and binary data. We used a CVS-system to deploy our documentation and source code to a Linux server at NTNU. This has helped us a lot since we have been working both at home and at the university.

[1] http://www.eclipse.org/
[2] http://texlipse.sourceforge.net/
[3] http://eclipseme.org/
[4] http://www.teaminabox.co.uk/downloads/metrics/
[5] http://www.modelistic.com/
[6] http://java.sun.com/products/sjwtoolkit/
[7] http://www.miktex.org

CHAPTER 5

Requirements elicitation and process

This project aims to develop two applications in order to test the Peer2Me framework. Before development can start, we need to create a requirements specification. Part IV and V each contain a chapter called "Requirements". The "Requirements" chapter consists of four sections: "Scenario", "Goals", "Functional requirements", "Use case diagram" and "Quality requirements". This chapter explains how these sections are constructed.

5.1 Scenario

We start the process by creating a scenario we find interesting. This scenario is easily understood, even by people with very little technical knowledge. The scenario description also helps the reader to understand what we want to accomplish in more general terms.

5.2 Goals

The goals listed in this section only focuses on the goals of the specific application. By examining the scenario more closely and from a technical standpoint we can extract the goals we need for the application. These goals are the fundament of the functional and quality requirements.

5.3 Functional requirements

There are several ways of extracting functional requirements from the goals, such as the Sutclife technique used by Sars and Norum in [22]. We will not apply this technique, our own programming experience to find these requirements .

The functional requirements are listed in a table with a given ID for each requirement. Each requirement only contains a specific requirement, not several. If it is not possible to describe a requirement using only one non-complicated sentence, the requirement is split into "sub requirements". This results in the following structure: 1 -> 1, 1.1, 1.2, 1.3 and so on. We will keep in mind that the main purpose of developing these applications is to test the Peer2Me framework.

5.4 Use case diagram

In order to explain how the functional requirements relates to each other and the users, we have chosen to include a use case diagram of the application. It is a standard use case diagram with actors and use cases.

5.5 Quality requirements

There are three groups of quality attributes: Business, architecture and system quality. We will in this report only focus on the system quality. Quality requirements are non-functional requirements such as performance, modifiability, usability, testability, availability and security. A quality requirement can be presented as a scenario. A concrete scenario is most often derived from a more general scenario. We will only list concrete scenarios in our elicitation. A quality attribute scenario is illustrated in 5.1.

ID – Description of requirement	
Source of stimulus	The instance which invokes a specific action in the system. Often a user or developer.
Stimulus	Description of what action the source of stimulus has begun
Environment	Which environment the action takes place in. Examples of environments are: runtime and designtime
Artefact	The artifact that is affected by the stimulus
Response	How does the system respond to the stimulus?
Response Measure	A specific measurement that can relate to the response. This measurement can be given in time, lines of code, number of bytes etc.

Table 5.1: *Description of quality attribute scenario [5]*

Availability is one of the most important quality attribute in a mobile peer-to-peer network and it is important to remember that connection with peers can be easily lost in a mobile environment. We have therefore chosen to pay special attention to the availability scenarios. Also, after having tried the test-applications made by Sars and Norum [22], we were quite disappointed with the user interface. It is quite obvious that GUI had not been a high priority, we claim that in order for a user to start using a mobile application, it has to be user friendly. This may not have been the case 6 years ago, but today the focus is very high on usability. We will therefore see if it is possible to develop a user friendly application using the Peer2Me framework.

Part III

Prestudy

CHAPTER 6

Essential Cooperation Theory and Technology

This chapter will explain the central concepts about peer-to-peer (P2P) systems and how they can be used on mobile devices without the use of an infrastructure. Also, some basic information about groupware and Computer Supported Cooperative Work (CSCW) is explained in order for the reader to understand the challenges we are faced with.

6.1 Peer-to-peer

Peer2Me is a based on a peer-to-peer (P2P) architecture. Computer systems can be either centralized or distributed. Workstations and mainframes are examples of centralized systems. Distributed systems are classified into two more architectures; the classical client-server model and the P2P model, see Figure 6.1.

The definition of P2P by Wikipedia is:

> "A peer-to-peer (or P2P) computer network is a network that relies on the computing power and bandwidth of the participants in the network rather than concentrating it in a relatively few servers. P2P networks are typically used for connecting nodes via largely ad-hoc connections" [42]

Lund and Norum [22] has recognized the following advantages of P2P networking:

- **Capacity:** Since a network connection between two peers does not depend on an central server, bandwidth, storage and processing power on the edge of the network is better utilized.
- **Independency:** Does not depend on a central server.
- **Configuration:** All peers are equal in terms of functionality and role. The network becomes self-configurable.
- **Decentralization:** In contrast to the client-server architecture, the P2P architecture are more decentralized. There is no significant load on one particular peer that creates a bottleneck in the network.
- **Extensibility:** Peers can easily join the network and increase its value.

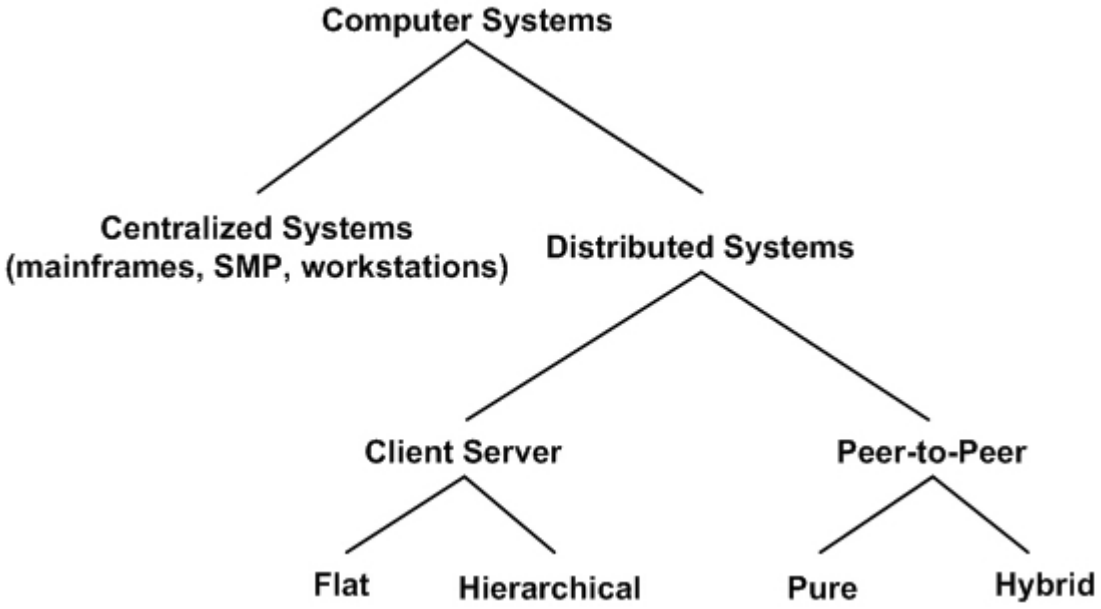


Figure 6.1: *Taxonomy of computer systems [37]*

- **Fault tolerance:** No single point of failure.

The resources are shared between peers, and a peer can either be a server or a client. In a P2P system, one peer can be connected to many other peers, or just one. This creates a solid and flexible architecture. There are three kinds of P2P networks [42]:

Pure P2P - All peers can act as clients and servers, there is no central server or router.

Hybrid P2P - Has a central server which keeps information on peers and responds to requests for that information.

Mixed P2P - A combination of pure and hybrid P2P.

Table 6.1 compares the pure and hybrid P2P networks.

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

Table 6.1: *Comparison of pure and hybrid model*

There are several applications available on the Internet that use P2P as an infrastructure for networking. Most popular are filesharing applications such as KaZaA and Direct Connect. Such applications creates a major challenge when it comes to digital content rights. Instant messaging applications such as MSN Messenger, Yahoo Messenger and ICQ are also quite popular. It is important to remember that applications based on P2P networking has very high network externalities. As more people start using them, their value increases drastically. It is much more convenient to have thirty people on your contact list in MSN Messenger than one.

6.2 Mobile P2P networks

Mobile P2P networks should be based on a pure P2P model so that the peers can discover services without the use of a central node. There are two different ways of implementing such a network:

- **Without an infrastructure:** Networking by using Bluetooth or Infrared.
- **With an infrastructure:** Networking by using GSM or UMTS.

Mobile P2P surely comes with a significant amount of challenges. Although the article is old, Forman and Zahorjan has identified three categories of challenges for mobile computing which are highly relevant for mobile P2P systems [12]. The following list briefly address the topics within these categories.

- **Communication:** Varying bandwidth, interference, security, users outside coverage area, frequent disconnections, delays and integration of heterogeneous network
- **Mobility:** Difficult to address devices and location-dependent.
- **Portability:** Size, weight, limited battery power, exposed to weather, must operate in noisy surroundings

When designing P2P applications for mobile devices, it is very important to remember that the devices are constantly moving around and the connection may be lost at any time. This can lead to incomplete data transfers between peers which must be accounted for in order to create solid and useful applications.

6.3 Mobile Ad Hoc Networks (MANET)

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

> "A personal area network (PAN) is a computer network used for communication among computer devices (including telephones and personal digital assistants) close to one person. The devices may or may not belong to the person in question. The reach of a PAN is typically a few meters. PANs can be used for communication among the personal devices themselves (interpersonal communication), or for connecting to a higher level network and the Internet (an uplink). Personal area networks may be wired with computer buses such as USB and FireWire. A wireless personal area network (WPAN) can also be made possible with network technologies such as IrDA and Bluetooth." [43].

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

> "A mobile ad-hoc network (MANET) is a self-configuring network of mobile routers (and associated hosts) connected by wireless links the union of which form an arbitrary topology. The routers are free to move randomly and organize themselves arbitrarily; thus, the network's wireless topology may change rapidly and unpredictably. Such a network may operate in a standalone fashion, or may be connected to the larger Internet" [39].

Cellular telephony relies on systems such as GSM and UMTS, which rely on infrastructure such as base stations. Ad hoc mobile networks do not need such infrastructure, which has its' advantages. [13] lists three of these advantages:

- **No infrastructure required:** Since no infrastructure is required, the network can be deployed spontaneously when needed anywhere.
- **Self-organization:** In a wired network, the topology is determined by the cables that connect the nodes to each other. In an MANET, the network is created as soon as two nodes are within each other's PAN. This means that the network is continuously reconfigured as mobile devices enters and exits each other's PANs.
- **Fault tolerance:** Since there are no infrastructure, the mobile devices cannot fail because of a base station, which is the case in networks that rely on GSM communication. The only way a MANET can fail is if one node fails, but since the network is based on P2P communication, the network can easily be self-reconfigured.

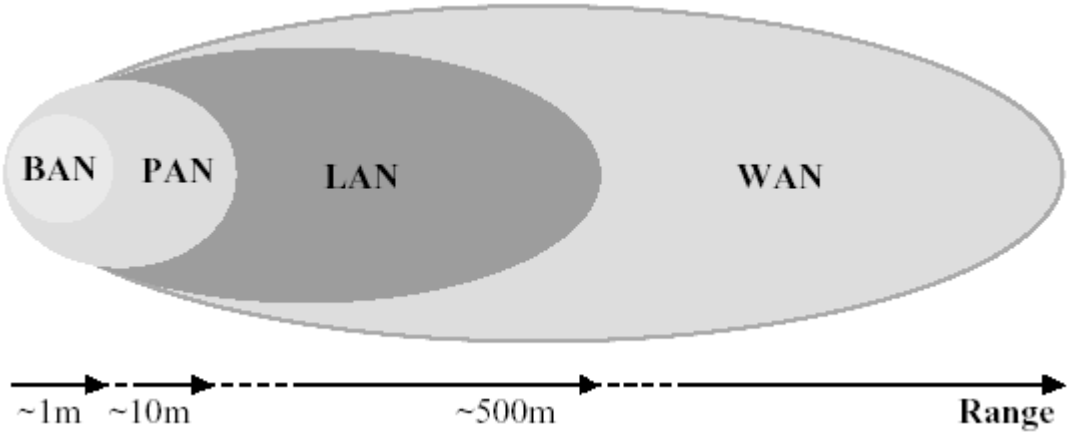


Figure 6.2: *Ad hoc networks taxonomy [10]*

Figure 6.2 illustrates the taxonomy of ad hoc networks. The horizontal axis shows the reach of each classification. The classifications are Body(BAN), Personal(PAN), Local(LAN) and Wide Area Networks(WAN). The Peer2Me framework is currently being used in PANs because it is based on the Bluetooth technology. WANs cover a much larger area such as a campus or a part of the city.

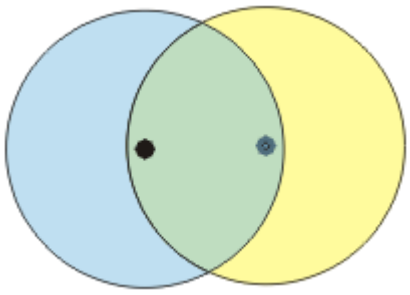


Figure 6.3: *A singlehop ad hoc network [15]*

Figure 6.3 illustrates a singlehop ad hoc network where only two peers are involved. The communication is performed directly between the two peers. Several nodes can make up a multihop ad hoc network

where some nodes are out of direct contact with each other. These communication between these nodes are then forwarded by the other intermediate nodes in the network. Figure 6.4 illustrates the multihop where the black lines are the communication paths.

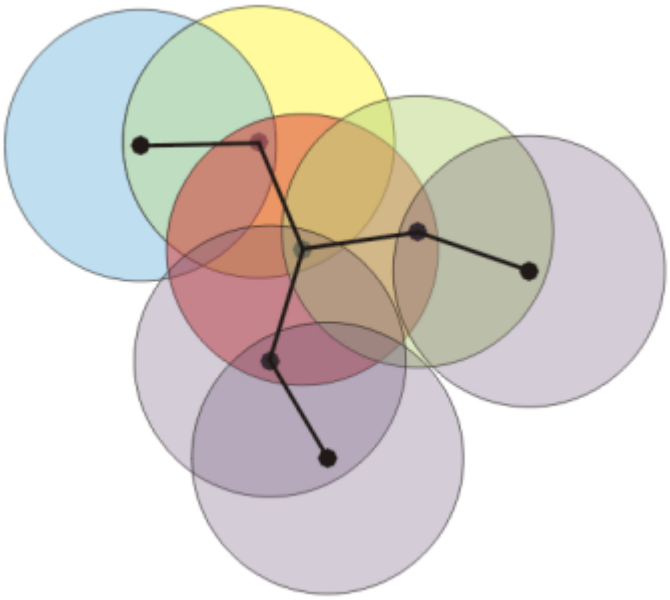


Figure 6.4: *A multihop ad hoc network [15]*

Peer2Me uses a form of PAN to communicate. A Bluetooth PAN can consist of up to eight devices. One device must function as a master, while the other seven must function as slaves. This is called a piconet. If there exists two piconets and one of the devices are connect to both piconets, a scatternet has been established, see Figure 6.5.

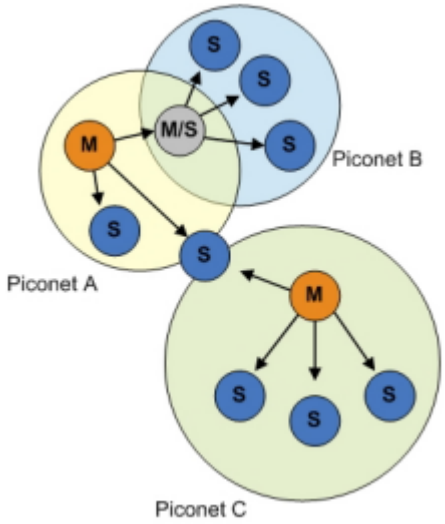


Figure 6.5: *Scatternet comprising three piconets [23]. m = master, s = slave, m/s = slave and master. The three piconets A, B and C together form a scatternet.*

However, this means that a message from one node may have to be forwarded by several nodes, which raises security concerns. The problem with the current Bluetooth implementation is that it does not allow a node to function as both a master and a slave simultaneously. This means that scatternet implementation of Peer2Me will have to wait.

6.4 Communication and Collaboration

Communication and collaboration among humans is a natural part of our civilization. Most of us change our behaviour in different contexts and we rarely give it much thought. It sort of happens automatically. The way we interact with other people are becoming more technologically dependent. Communication has evolved from regular face-to-face communication and letters to mobile phones and e-mails. Much research has been done in analysing how humans collaborate without technical devices in order to create the same context with the use of computers, hand held devices and other communication tools. It has proven to be very hard to transfer the "real world" to the "digital world". Consider the following scenario which takes place in an office building:

> Jamie takes a break from his computer and takes a walk to the coffee machine to get a warm drink. There he meets his colleague Adam, who looks quite nervous. Jamie remembers that Adam is going to present project to some investors this day and starts talking to him. Adam tells Jamie that he is not sure if the investors will like his idea and has a quick discussion with Jamie. Jamie calms him down and give Adam a couple of tips on how he should present the material. Later that day, Adam holds a successful presentation.

This scenario is not easily transferred to the digital world. Just look at all the different elements in the scenario: Chance encounter at the coffee machine, Adam's physical appearance triggers Jamie's supporting talk, the discussion itself, Jamie's calming voice etc.

6.4.1 Groupware and Computer Supported Cooperative Work (CSCW)

In order to transfer scenarios similar to the one described above to the digital world, much research in the field of CSCW has been undertaken. Unplanned, informal cooperation is very often more effective and useful than planned meetings. When people start working together outside a formal setting, no

special protocol is enforced which sets out a creative atmosphere. The study of CSCW also includes formal settings such as scheduled video conferences. Groupware are computerized systems that have been created to support CSCW. These systems can be classified in a time-space matrix illustrated in Table 6.2.

	Same Time	**Different Times**
Same Place	face -to-face Interaction	asynchronous Interaction
Different Places	synchronous distributed Interaction	asynchronous distributed Interaction

Table 6.2: *Groupware Time Space Matrix [11]*

We can further categorize groupware in two more classes: Mobile - and non-mobile groupware. Non-mobile groupware is the most common type. These applications run on systems that you normally do not carry around while in use such as desktops and laptops. A laptop may be portable and used everywhere, but the user is seldom moving around when using the system. Mobile phones and PDAs however are the target for mobile groupware which Peer2Me is focusing on. Since Peer2Me uses Bluetooth as means of communication, the framework is designed for applications in the same place, but both real and asynchronous time space.

Since Peer2Me uses some kind of PAN, peers needs to be in close range of each other such as face-to-face. These encounters can either be planned or unplanned. A PAN creates a digital sphere around a person and when two of these spheres comes within range of each other, collaborative applications can be run, see Figure 6.6.

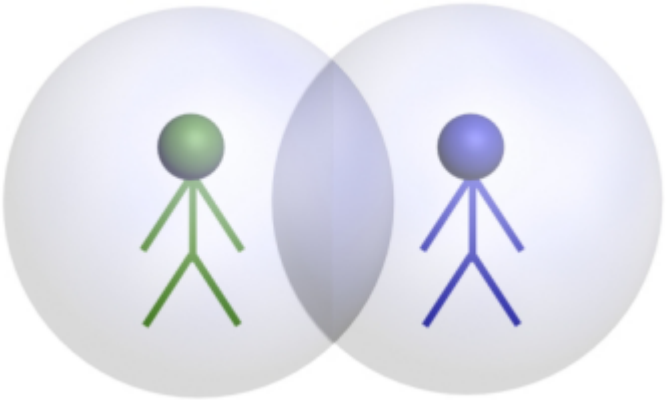


Figure 6.6: *Digital spheres intersecting*

Also, Peer2Me does not rely on any infrastructure which opens the possibility for a diverse range of useful P2P applications. In [13], different types of applications are proposed:

- Alert the user of friends that are nearby in crowded places.
- Identify people we want to meet
- Match making algorithm that takes into account our preferences and interests
- Spread rumours
- Exchange of personal data

- More complex tasks such as trade delivery tasks

Many of these applications are suitable in impromptu collaboration. In [22], Norum and Sars identified three categories of such applications:

- **Requiring user interaction:** The user has to trigger the application.
- **Automatic collaboration:** The application collaborates with other users on its own by running in the background.
- **Automatically triggered collaboration:** The application searches and locates other peers and the alerts the user which has to act accordingly.

CHAPTER 7

Peer2Me

The Peer2Me framework is the result of a master thesis by Carl-Henrik Wolf Lund and Michael Sars Norum, and is a project related to the MOWAHS project Mobile Work Across heterogeneous Systems. In this chapter we will give a description of Peer2Me and how to use it for mobile application development

7.1 Introduction

The Peer2Me framework is a framework for developing communicating applications on mobile phones and handheld devices, enabling users to communicate in an ad-hoc environment. The Peer2Me framework is based on the distributed system concept, Peer-to-peer (P2P).

Peer2Me is can be classified as a hybrid P2P model (see Chapter 6.1) since implementation on mobile phones has shown that a device cannot function both as a server and a client at the same time using the Bluetooth technology. Furthermore, the Bluetooth API for mobile phones does only support connection to seven devices simultaneously. The creators of Peer2Me initially wanted it to support a pure P2P model, but this is currently very hard to accomplish.

7.2 Design

This Section will describe the design of Peer2Me.

7.2.1 Domain Concepts

In order to understand how the system works, some terms need to be explained:

- **Framework:** The core entity of the framework. Manages the resources such as known peers and available network mediums. It is also the interface between the application and the rest of the system.
- **Node:** A node is a peer; a mobile phone running the framework. A node must be either a slave or a master, not both.
- **Network:** Abstraction of the network layer. Is indirectly access through the framework instance.

- **Service:** An application must be running a specific service in order for other peers to locate the application and connect to the respective peer.
- **Group:** A collection of nodes running the same service. The group must have one master node.
- **Message:** Messages are exchange between the different nodes within a group in order to communicate.
- **Application:** Software that uses the framework.

Figure 7.1 illustrates a conceptual model of how the Peer2Me framework works. A mobile phone is a node which runs an application with a specific service ID. This node then communicates with other nodes running the same service by messages which is sent using the framework. The network technology used is Bluetooth.

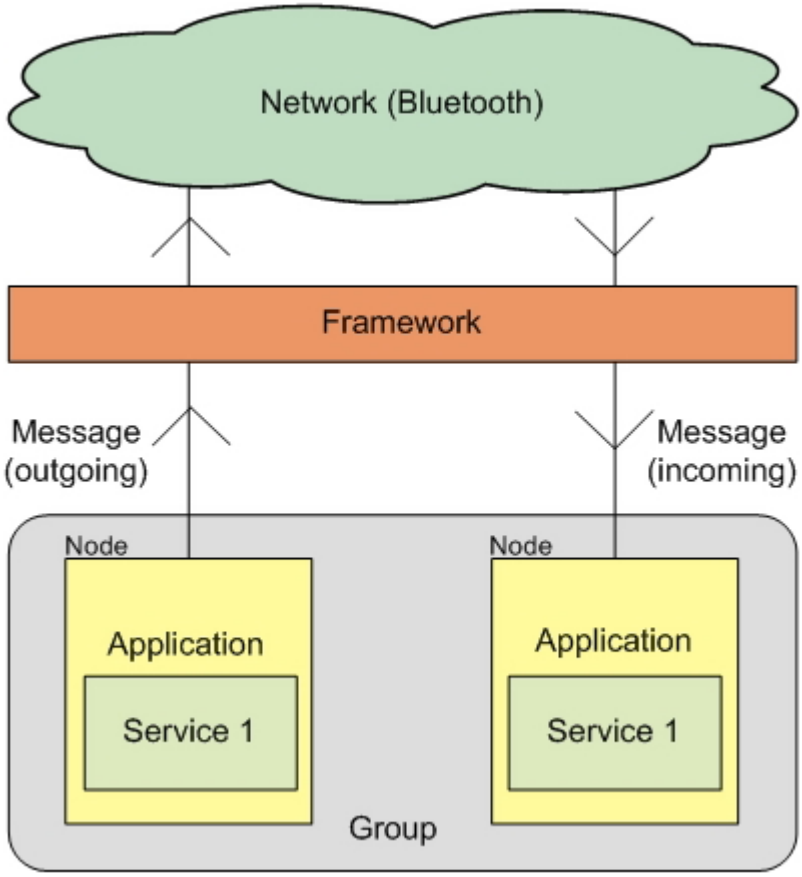


Figure 7.1: *Domain concept of peer2me*

In order to make two or more nodes communicate, they have to be a "member" of the same group. The Peer2Me framework is built upon the core J2ME components, the CLDC(Connected, Limited Device Configuration) and the MIDP(Mobile Information Device Profile). The MIDP must be version 2.0 or later. Peer2Me consists of four main modules, the framework itself, the network interface, the Bluetooth module and the domain module. Figure 7.2 illustrates a layered architecture of Peer2Me, the J2ME core components and a Midlet.

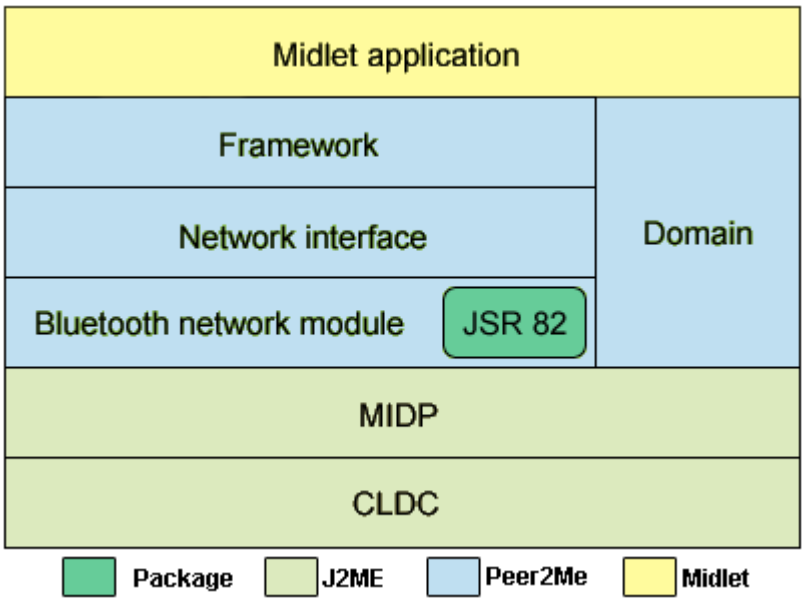


Figure 7.2: *Layered architectural overview*

The generic network interface enables the developer to control the technology specific network modules. In the current version of Peer2Me, the network technology is Bluetooth. The idea is that new network technology modules can easily be added at a later point and the same network interface can be used in order to access it. The Bluetooth network module uses the JAWBT package called JSR82, see [26] for more information. The package is not a core J2ME component, but is optional.

The Peer2Me framework is available as a jar-file. The classes in the jar-file is organized in a package structure. There are five main packages:

- *no.ntnu.idi.mowahs.project.framework*
- *no.ntnu.idi.mowahs.project.bluetooth*
- *no.ntnu.idi.mowahs.project.domain*
- *no.ntnu.idi.mowahs.project.network*
- *no.ntnu.idi.mowahs.project.util*

The usage area of the first four packages are illustrated in Figure 7.1 and Figure 7.2. The util package contains "persistence" classes that can be used to access the recordstore on mobile phones. This package is only used by MIDlets that requires access to the recordstore[1].

As a curiosity, we have chosen to include Table 7.1 which shows some code statistics of the Peer2Me framework. It only shows statistics from the Peer2Me framework and not the test-applications Norum and Sars developed.

[1] A J2ME compatible mobile phone can store application data in a recordstore. The recordstore is a persistent storage which MIDlets can store and retrieve data at any time

Aspect:	Value:
Lines of code:	1163
Number of classes:	26
Number of interfaces	6
Number of packages:	18
Maximum inheritance tree depth:	6

Table 7.1: *Framework statistics*

CHAPTER 8

State of the art

There are currently several projects that are similar or related to Peer2Me. This chapter presents some of the most interesting projects. A couple of them have been mentioned in the master thesis by Sars and Norum [22], but some changes has occurred since they were last reviewed.

8.1 Bluecove

Bluecove is an open source implementation of the JSR 82 Bluetooth API for Java. Bluecove is a library that can be used to connect a PC with a Bluetooth dongle and a mobile phone with JSR 82 support together in a Java environment. The library can be downloaded from SourceForge [14]. The current version of Bluecove only supports the Windows XP SP2 Bluetooth stack. However, the goal is to support other operating systems and other Windows stacks. Ben Hui has created a guide that explains how to connect a PC and a phone together using Bluecove. The guide can be found at [16].We have tested Bluecove using this guide with positive results.

Evaluation

Peer2Me is meant for ad hoc connection between mobile phones only, not between a PC and a mobile phone. Bluecove does not allow for connection between mobile phones. This means that the two libraries may overlap each other, creating a robust framework that can connect many mobile phones and PCs together in an ad hoc fashion. However, Bluecove will only work on a PC that uses the Windows XP SP2 Bluetooth stack. As optimists, we tried using a different stack to see if we could get it working, but that experiment failed.

8.2 Bluetooth Ad-hoc Networking for Inter-Vehicle Communication

The Communications Research Group at University of Sussex has investigated the area of ad hoc communication between vehicles passing by each other in traffic [32]. The project proposes the use of Bluetooth for this purpose. If two vehicles are driving in the same direction with speeds 97 km/h and 113 km/h they will be in range of each other in a duration of 44 seconds. During this time, much information can

be exchanged between these vehicles using Bluetooth. These calculations presumes a Bluetooth range of 100 meters, which is the theoretical maximum coverage range a Bluetooth device has. Many piconets can be created and form a large MANET. There is sadly not much information about the project, other than it is under investigation. The projects web page was last updated 14th of December 2004.

Evaluation

The project seems very interesting, but the usage area is slightly different than what the Peer2Me framework is meant for. Also, the documentation is poor and no new information has been published lately.

8.3 Umbrella.net

Umbrella.net is a developing research platform at the Networks and Telecommunications Research Group at Trinity College Dublin [9]. The platform explores the mobile ad hoc network possibilities in public and urban spaces using an umbrella that is connected to a PDA with the Umbrella software. The aim of the project is to connect individuals, both strangers and friends together in an ad hoc environment. The PDA that is connected to the umbrella starts a service discovery when the umbrella is opened and connects to other people that also have their umbrellas open. This might seem a bit silly, but is in fact very interesting. People having the same needs (protection from the rain) get connected wirelessly and can then share information. This information can be anything from chatting to a sudden market were people can buy and sell from each other. Nodes do not need to be in direct connect with each other, which means that information can pass through a lot of nodes before reaching the destination node.

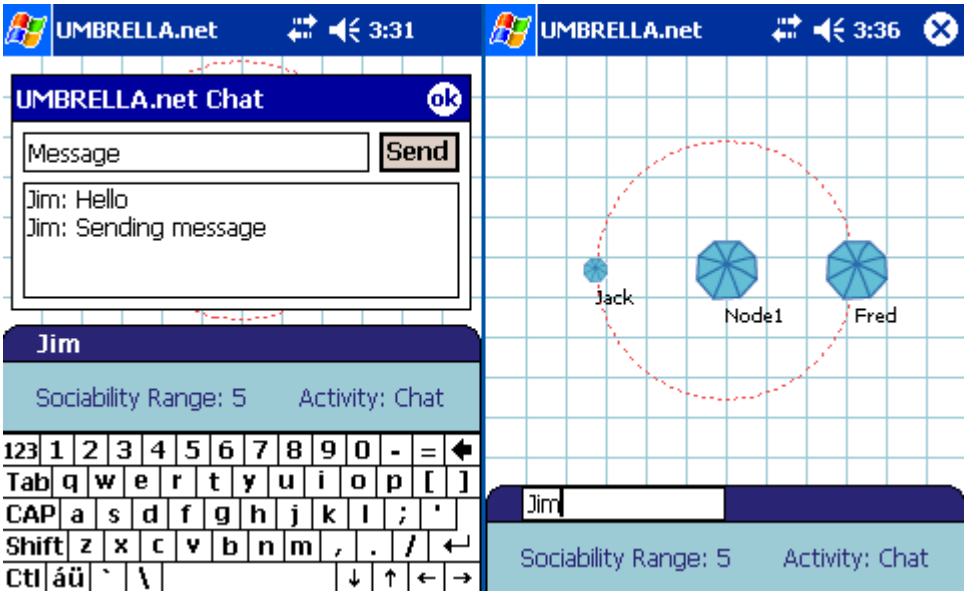


Figure 8.1: *The GUI of the Umbrella software [8]*

Evaluation

Umbrella.net seems to have the same goals as Peer2Me, but the projects target platform is a PDA running Windows CE. The software is not available for download and there is no information regarding programming platform or architecture. The graphical user interface of Umbrella is very good and seems intuitive and easy to use. The project has received high recognition and there have been several events

and exhibition around the Umbrella.net project in countries like Ireland, Switzerland, USA, UK and Australia.

8.4 BEDD

BEDD is mentioned in the master thesis by Sars and Norum [22], but a few things have changed since last review. BEDD is a software suit developed and maintained by the BEDD Corporation [6] that enables mobile phones to communicate ad hoc. The software can be downloaded for free from BEDD's website. It is also free to distribute. The software runs in the background on your mobile phone and automatically searches for other phones running the BEDD software. BEDD is platform independent and uses Bluetooth as network medium. Among the applications in the suit we find:

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

BEDDbay - Place ads and view other ads. Buy and sell

BEDDtalk - Chat application

BEDDfish - Allows the user to send free text messages to Bluetooth devices that does not have BEDD installed

BEDDbuddies - Get alerted when friends are within range

BEDDings - Sounds, alerts, skins etc

BEDDbios - Your profile viewable for others to see

BEDDshare - Allows the user to share the BEDD software

Evaluation

After the work of Sars and Norum [22] was completed, the BEDD software became free for everyone to use. The software relates very closely to Peer2Me. The difference is that BEDD is an end user application and not an open development platform. The applications available in the BEDD software suit are the exactly the kind of applications that can be developed using J2ME with Peer2Me.

8.5 Other projects

The following projects are mentioned in [22] and we will here just quickly review them.

JXTA - This is peer-to-peer technology developed by Sun Microsystems that enables developers to create distributed computing software. JXTA is platform independent and can connect several different devices together in an ad hoc fashion. The JXTA itself is too heavyweight to be deployed on mobile phones, so Sun created JXME for this purpose. JXTA uses TCP/IP for communication. See [24] for more information.

JSR-259 (Ad Hoc Network API) - A library for J2ME that enables communication between mobile devices in a peer-to-peer ad-hoc network environment. The project is still in progress and the final approval ballot is set to the end of fourth quarter 2005. The JSR API is said to include methods for:

- Service discovery
- Service registration
- Service availability alert

- Service and service capability inquiry
- Remote service consumption

For more information, see Sun's JSR specification website, [25].

OBEX - Object Exchange Protocol. The protocol is maintained by the Infrared Data Association. OBEX was designed to allow exchange of binary dataobjects over an infrared link. It has also been adopted by Bluetooth SIG and the SyncML wing of the Open Mobile Alliance (OMA). In contrast to HTTP, OBEX is stateless. More information on OBEX can be found on Wikipedia [40].

SyncML - The former name of OMA. SyncML allows synchronization of contact and calendar information between a handheld device and a computer. Current version of SyncML is 1.1.2. More information on SyncML and OMA can be found on Wikipedia, see [44] and [41].

CHAPTER 9

Technology

This chapter contains information about the latest technologies related to ad hoc networking and mobile phones. Since this project involves creating applications on mobile phones, an investigation in todays mobile phone has been done. Peer2Me was developed using Java 2 Micro Edition (J2ME) and a the latest information available has been retrieved. Bluetooth has been used as network technology in the current version of Peer2Me. There are several other wireless network technologies worth investigating in addition to Bluetooth. Wireless devices have limited bandwidth which must be considered when developing applications for such devices. XML enables information to be exchanged and interpreted regardless of transport medium and application language. The problem is that with XML comes overhead. XML technology has been included in this chapter as it is used in the applications described in Part IV and Part V.

9.1 Mobile phones

There are plenty of mobile phones available that supports J2ME, so listing all of them is pointless. Applications on mobile phones need to have high usability in order to catch the end-user attention. Our applications will use more graphical items than the applications developed by Sars and Norum [22] and it is therefore necessary to find the optimal screen resolution to adapt to. We have therefore created a table which contains the different screen resolutions on mobile phones from different manufacturers, see Table 9.1.

Nokia 6021 is the phone that has the lowest screen resolution among the mobile phones considered. The applications to be developed in this project must "fit" on such small displays. Since J2ME applications should be tested on real phones and not just emulators, NTNU has supplied us with the following two mobile phones:

- **Sony Ericsson p900:** This phone runs the Symbian OS 7.0. It supports applications created in Java, C++ and Visual Basic [1]. This phone has a faulty J2ME implementation which was discovered in [22]. It is possible to develop J2ME applications for this phone, but some source code in Peer2Me has to changed in order for the application to work.
- **Siemens S65:** This phone supports also supports J2ME MIDlets and we have not found any errors

[1] Sony Ericsson S700i and Qtek S110 are the phones with the largest screen resolution
[2] Nokia 6021 is phone with the smallest screen resolution

Manufacturer	Model	Resolution
Sony Ericsson	P900	208 x 320
Sony Ericsson	W800i	176 x 220
Sony Ericsson	K750i	176 x 220
Sony Ericsson	S700i [1]	240 x 320
Sony Ericsson	W550i	176 x 220
Sony Ericsson	K700i	176 x 220
Sony Ericsson	K600i	176 x 220
Siemens	S65	132 x 176
Siemens	M75	132 x 176
Samsung	D500	176 x 220
Samsung	Z500	176 x 220
Samsung	E730	176 x 220
Samsung	E530	176 x 220
Samsung	E720	176 x 220
Nokia	8800	208 x 208
Nokia	6230i	208 x 208
Nokia	6021 [2]	128 x 128
Motorola	V600	176 x 220
Qtek	S110 [1]	240 x 320

Table 9.1: *Display resolutions of mobile phone with J2ME support*

with the J2ME implementation yet. The only problem is that this phone does not support Java threads. More information about this phone can be found at [27].

In addition, we will use one of the group members mobile phone; a **Sony Ericsson w800i**[2]. This phone is the newest among the mobile phones at this time and Sony Ericsson has fixed the J2ME bug that were discovered on the P900. We will probably use the w800i and P900 the most. P900 works well as long as it is not the master node in a Bluetooth network. During the implementation process the applications will be tested on a couple of **Sony Ericsson k750i**, which some friends of us happen to have.

9.2 Java 2 Micro Edition

Since the resources on a mobile phone is limited compared to a desktop, neither the J2EE or J2SE could be used as development platform for mobile phones. Therefore, Sun developed the Java 2 Micro Edition so that Java applications could be made for consumer and embedded devices as well.

9.2.1 J2ME architecture

The J2ME architecture defines configurations, profiles and optional packages as elements for building complete Java runtime environments that meet the requirements for a wide range of devices. The J2ME architecture has been created with limited memory, processing power and I/O capabilities in mind. To implement in J2ME, a configuration composed of a virtual machine and a minimal set of class libraries are needed. There are currently two configurations available for J2ME: CDC (Connected Device Configuration) and CLDC (Connected Limited Device Configuration), see Figure 9.1. CDC is for devices that have more resources available such as PDAs and set-top boxes. CLDC is for mobile phones, which is the configuration that will be used in this project. There are currently two versions of the CLDC configuration: 1.0 (JSR 30) and 1.1 (JSR 139). CLDC 1.1 is a revised version of 1.0 and is said to be backwards compatible with 1.0.

We have tried to deploy a Midlet, compiled using CLDC 1.1, on several mobile phones without success. CLCD 1.0 will therefore be used in this project.

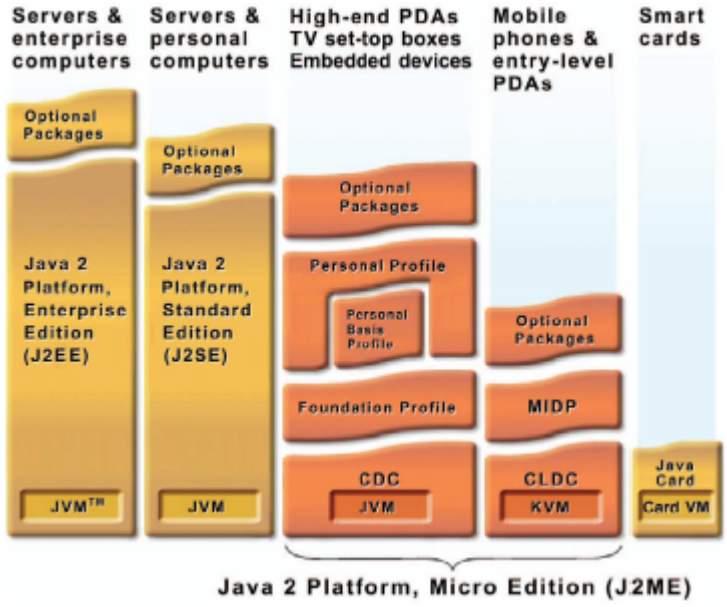


Figure 9.1: *Overview of the Java environments [17]*

A profile is needed to define the life cycle, user interface and access to device specific properties. Sun has developed the profile, MIDP for this purpose. Current version of MIDP is 2.0. Optional packages that provides added functionality is also available, but some of them will not function on all phones even if they support J2ME and MIDP 2.0. For more information, read the J2ME whitepaper [17] and see Sun's website [18].

9.2.2 Optional packages

As mentioned, there are optional packages for J2ME, and the following is used in this project:

- **JSR 82** - The Java™APIs for Bluetooth. This is a specification that standardizes a set of Java APIs to allow Java-enabled devices to integrate into a Bluetooth environment. The Peer2Me framework depends on this package and must be imported in the J2ME MIDlets that imports Peer2Me.
- **JSR 75** - Contains two packages: One for accessing PIM (Personal Information Management) data and one for accessing the filesystem. This package is meant for PDAs and newer mobile phones. By having access to the device's PIM, a Midlet can extract calendar information, address book and other useful information from the phone. The package that accesses the filesystem is very useful and we see that it has huge potential. One of the test applications in this project uses this package, see Chapter 20 for more information.

9.3 Wireless network technologies

Several different wireless network technologies have been developed and are designed with specific usage areas in mind. This section describes and compares these technologies. Wireless data transfer is more vulnerable compared to wired data transfer. Theoretically, anyone can "listen" to wireless data transfers. Security is therefore important to ensure privacy, especially if sensitive information is transferred.

9.3.1 Bluetooth

Bluetooth is an industrial specification for wireless personal area networks (WPAN). Bluetooth provides a way to connect and exchange information between devices like PDA, mobile phones, computers etc. via a low cost, short range radio frequency (2,45GHz)
The range and power consumption are divided into three classes:

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

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

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

The most common is class 2, while a hybrid between class 1 and class 2 is often found in mobile phones and handheld units. This is a suitable tradeoff, range vs. power consumption.
Bluetooth did first arise as version 1.0, but is commonly known as version 1.1 or 1.2 which is most widely implemented. 2.0 is the latest version, and the greatest improvement in v2.0 is the introduction of *Enhanced Data Rate* which increases the speed to a maximum of 2.1Mbit/s. About three times faster than version 1.1 and 1.2.
Bluetooth also supports a set of *profiles*, each a description of the Bluetooth unit; a Bluetooth handsfree would use the *Headset*-profile, while a Bluetooth mouse would use the *HID*-profile, *Human Interface Device*, and so on.
Bluetooth will continue to evolve, and The Bluetooth Special Interest Group (SIG) [36] controls the development.

Security

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

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

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

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

Since Bluetooth does not provide end-to-end security, it is exposed to several security issues. These are discussed in a document published by Adam and Ben Laurie in the article "Serious flaws in Bluetooth security lead to disclosure of personal data" [21].

9.3.2 Zigbee

Zigbee is a set of high level protocols using a small, low powered digital radio based on the IEEE 802.15.4 specification. Zigbee is designed to be simpler and cheaper than other Wireless PANs, such as Bluetooth, but is still more expensive due to the lower demand and therefore higher production costs.
Zigbee is designed as a three layer solution;physical layer, medium access control layer and data link layer. It can operate on three different frequencies depending on the situation. 2.4 GHz, 915 MHz and 868 MHz, and each frequency has its own bandwidth limitation reaching from 250 Kbit/s per channel in the 2.4 GHz band to 20 Kbit/s in the 868 MHz band. The transmission rate is between 10 and 75 m.

Security

The security in the Zigbee standard are design to offer a high level of protection against attacks and are divided into four main services:

Data encryption: provides symmetric and asymmetric 128-bit using the AES - advanced encryption standards

Access control: the device using Zigbee maintains a list of trusted devices within the network

Frame integrity: protects data from being modified by parties without cryptographic keys

Sequential Freshness: reject data frames that have been replayed. The network controller compares the freshness value with the last known value from the device and rejects it if the freshness value has not been updated to a new value

The security features of ZigBee is discussed in "Industrial-strength security for ZigBee: The case for public-key cryptography" [7], an article written by Mitch Blaser.

9.3.3 Radio Frequency IDentification

Radio Frequency IDentification, RFID, is an automatic identification method, relying on storing and remotely retrieving data using devices called RFID tags or transponders. An RFID tag is a small object that can be attached to or incorporated into a product, animal, or person.
RFID tags is divided into three groups:

Passive: This type have no internal power source, but induces enough power through the radio frequency to transmit a response. The lack of onboard power supply means the amount of data that can be sent before loss of power is very small, but this makes it possible to make tiny tags.

Semi-active: This is very similar to the passive tag, but includes a small battery which gives a small amount of power to the chip. This means the tag can be optimized for sending data, without have to focus on inducing power, which again results in faster transfers.

Active: Also called *beacons*, uses a fixed internal power supply, which gives larger storage capacity and greater range. Many active tags have practical ranges of tens of metres, and a battery life of up to 10 years, and to rationalize the power consumption, the beacons often operate in intervals.

The purpose of an RFID system is to enable data to be transmitted by a mobile device, called a tag, which is read by an RFID reader and processed according to the needs of a particular application. The data transmitted by the tag may provide identification or location information, or specifics about the product tagged, such as price, color, date of purchase, etc.

9.3.4 Wireless Local Area Network

WLAN, short for Wireless Local Area Network, is the trademark for a set of product compatibility standards for WLANs introduced in 1997, and is intended to allow mobile devices communicate, get Internet access or connect to local area networks. WiFi is also used for wireless Voice over IP phones (VoIP).
Today, wireless networks is one of the most common wireless technologies, and are divided into four standards with different operating frequencies and bandwidth as shown in Table 9.2.

Specification	Speed	Frequency	Compatibility
802.11b	11 Mb/s	2.4 GHz	b
802.11g	54 Mb/s	2.4 GHz	b and g
802.11a	54 Mb/s	5 GHz	a
802.11n	100 Mb/s	2.4 GHz	b,g and n

Table 9.2: *The different WLAN standards*

802.11b was the first standard to be released, and got well implemented and quite common before 802.11g was introduced roughly one year later. The speed got nearly five times faster and the coverage area got wider. As a descendant to 802.11g, IEEE recently introduced the new 802.11n standard which again delivers higher speed and even wider coverage. All of these standards is operating on the commercial 2.4GHz frequency, which therefore is most common, but most exposed to interference. As a result of this, 802.11a got designed as an alternative operating on a higher frequency band, 5GHz. This band has less interference, but is more restricted. 802.11a has also a smaller coverage area than the other standards.
WLAN implements two different types of infrastructure:

Ad hoc mode: The WLAN devices communicate directly with each other without an access point. Devices within range of each other can connect and act like peers.

Infrastructure mode: This mode incorporates the WLAN device into a wired network via an access point.

Security

WLAN transmissions can either be open or secure. The security is implemented by encrypting the transmission using different methods. When the transmission is secure, one cannot automatically access the transmission in opposite to if the transmission were open.
The encrypting methods available:

WEP: Wired Equivalent Privacy uses a shared key and encodes the data that is being transmitted over air. WEP implements three levels of security using either a 40bit, 104bit or 232bit key, hence the strength of the encryption. This is often referred to as 64,128 or 256bit encryption. It uses the RC4 encryption algorithm and TKIP (Temporary Key Integrity Protocol).

WPA: Wi-Fi Protected Access was designed to deal with several weaknesses in WEP as an intermediate solution before WPA2. It does not support older access points or WLAN-hardware.
Instead of a fixed length key, WPA uses a passphrase. WPA is based on WEP, and uses both RC4 and TKIP. Unfortunately, this can be a security flaw.

WPA2: Wi-Fi Protected Access version 2, often referred to as 802.11i, is avoiding the security flaws in WEP and is based on the concept of a Robust Security Network (RSN) [3], which supports additional functionality. Hence, new equipment is required.

WEP contains security risks like jamming, insertion attacks and interception of data, while WPA2 is the best way to add security to a Wireless LAN, but the equipment must support WPA2. Newer equipment supports WPA2.

9.3.5 Comparison

The wireless technologies mentioned in the sections above are all solutions which can be used in a mobile ad hoc situation, but they have different qualities, where speed, range and power consumption are the most important. A simple comparison is given in Figure 9.2, 9.3 and 9.4. Note: all values are typical

values.

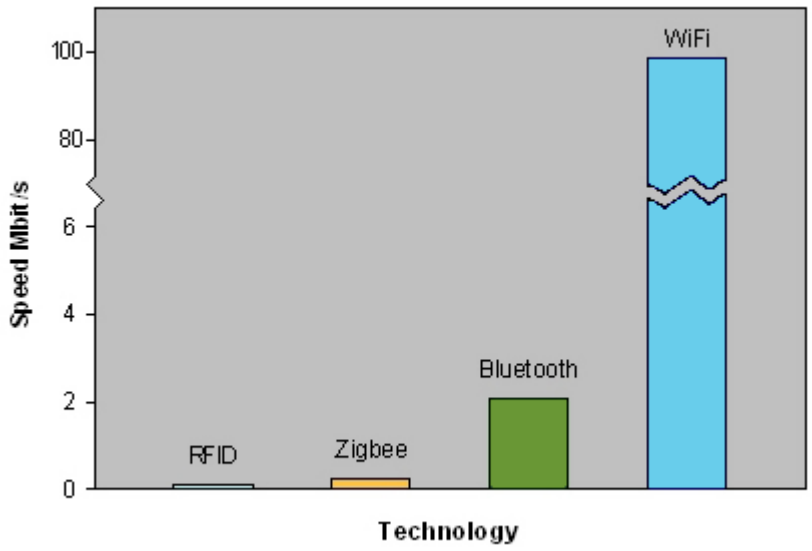


Figure 9.2: *Speed comparison*

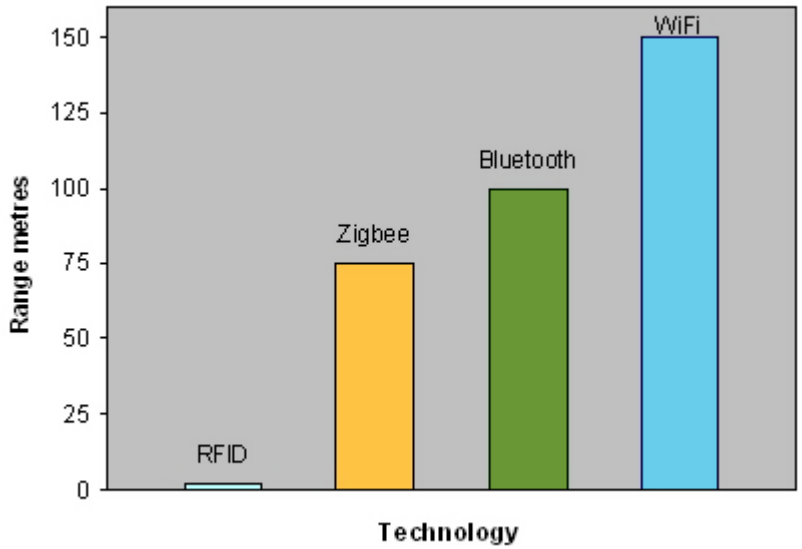


Figure 9.3: *Range comparison*

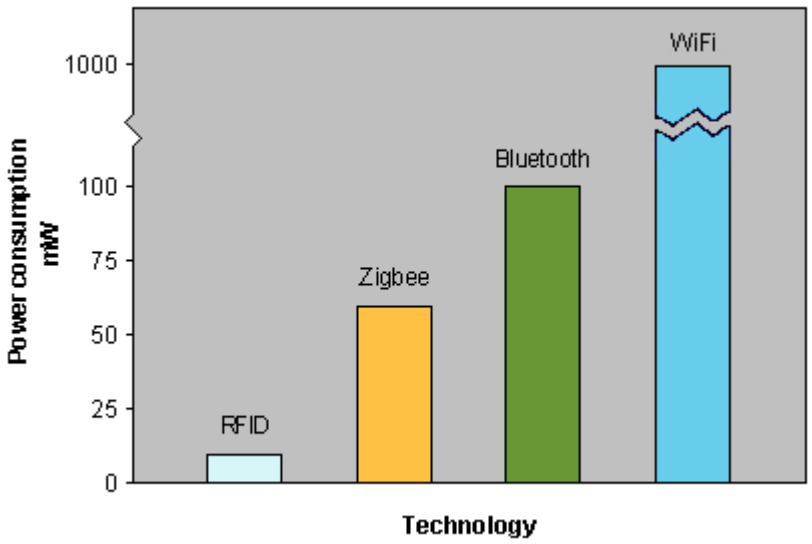


Figure 9.4: *Power consumption comparison*

As we can see from the figures, there is a correlation between speed, range and power consumption. Higher range means higher power consumption and higher range means greater speed.
Usually the design of new mobile units and software involves trade-offs, typically battery time versus performance, e.g. wireless coverage area versus power consumption.

9.4 XML

This section discusses briefly what XML is, its features and how it can be used. This section also describes the typical structure of an xml-file and software that can be used to process it.

XML stands for eXtensible Markup Language and is a tag-based document format to describe a set of information. XML makes it easy to store, retrieve and exchange information platform or software independent. XML can be compared to HTML, but XML can only represent information and is more general. The XML language has its strengths and weaknesses, as shown in Table 9.3:

9.4.1 Strengths and weaknesses

As we see in Table 9.3 XML is ineffective when it comes to space usage. This might be something you have to consider if the space available is limited, and instead you might have to used a simple "clean" text file. Nowadays with newer handheld units, the space limitation is not an issue. Therefore the applications we implemented uses XML for structuring data.

Strengths
Support for Unicode, allowing almost any information in any human language to be communicated
Self-documenting format that describes structure and field names as well as specific values
Hierarchical structure which is suitable for most types of documents
Platform-independent
Manifests as a plain text file
Weaknesses
Poor human readability
Ineffective use of space, tag-descriptions take up space
Can be some what redundant and can cause reduced application efficiency
Mapping XML documents to object oriented systems can be cumbersome
Text representation means huge files and no compression

Table 9.3: *Strength and weaknesses of XML*

9.4.2 Syntax

Xml files are built up of nodes, often called elements, sorted in a hierarchical order, where each element can have several attributes. This is shown in Listing 9.4.2 below:

Listing 9.1: *Example of an xmlfile*

```
1 <Rootnode>
2     <Node1>
3         <Node2 description="This is Node1's childnode">
4             <Node3 id="0" text="Text here"/>
5             <Node4 id="5" text="9"/>
6         </Node2>
7     </Node1>
8 </Rootnode>
```

An Xml file is built using *elements*, which can consist of either one or two *tags*. A tag is for example `<Node1>`, and when used in pairs as this: `<Node1>Sample text</Node1>` you can see the end tag with the / that corresponds to the starting. When creating an element with only one tag, it could look like this: `<Node1 id="5" content="This is me"/>`, with the slash at the end of the tag. **This is similar to HTML-syntax.**
As said, each element can have several attributes. Attributes are the fields "inside" the XML tag. As seen in Listing 9.4.2 line 4, this element have the attributes *id* and *text*. One important thing to remember is that you cannot have tags inside tags without them being elements.
As an example, this is not allowed:
`<Fish id="1" description="This fish is <b>BIG</b>">Trout</Fish>`. Here the `<b>BIG</b>` will be interpreted as an own element, and not as a bold typesetting when the data is extracted and used in HTML. To achieve this, XML accepts using some predeclared special characters, and the same XML element would look like this: `<Fish id="1" description="This fish is lt;bgt;BIGlt;/gt;>">Trout</Fish>`.
To ensure that the all tags are correct, all XML documents can be validated with an associated schema as described in Section 9.4.3 Validation.

9.4.3 Validation

An important feature of the XML concept is the ability to validate documents before processing them. This is done by comparing the xml document to a description document, called *schema*.
A schema describes the structure and constraints of the XML:

- defines elements that can appear in the document

- defines attributes that can appear in the document
- defines which elements are child elements
- defines the order of child elements
- defines the number of child elements
- defines whether an element is empty or can include text
- defines data types for elements and attributes
- defines default and fixed values for elements and attributes

There are different types of validation schemas, where DTD, XML Schema and Relax NG are the most common. DTD, Document Type Definition, is the oldest validation document and is being phased out. XML Schema, also called XSD, are the most common and is widely used due to its powerful and rich description of XML files. The drawback is that the specification in XDSs might be very large and complex and hard to understand and implement. Relax NG is the last validation document and has two formats, an XML and a non-XML based syntax. The biggest difference between XSD and Relax NG is that Relax NG is less complex and easier to read, but then again not as powerful.
A DTD associated with the xml examplecode given in Listing 9.4.2 is given in Listing 9.4.3.

Listing 9.2: *Example of an DTD file*

```
<!ELEMENT Node1 (Node2)>
<!ELEMENT Node2 (Node3, Node4)>
<!ATTLIST Node2 description CDATA #REQUIRED>
<!ELEMENT Node3 EMPTY>
<!ATTLIST Node3 id CDATA #REQUIREDtext CDATA #REQUIRED>
<!ELEMENT Node4 EMPTY>
<!ATTLIST Node4 id CDATA #REQUIRED text CDATA #REQUIRED>
<!ELEMENT Rootnode (Node1)>
<!ATTLIST Rootnode xmlns:xsi CDATA #IMPLIED
 xsi:noNamespaceSchemaLocation CDATA #IMPLIED>
```

As we can see in the DTD example, it consists of several element descriptions and something called *attlist*. *Node1* in the element description in the DTD has got only one element: *Node2*. This is because the Node2 is, and must be a child of Node1. Then the next element description, the description of *Node2* has got two elements: *Node3* and *Node4*. This is then because Node3 and Node4 are children of Node 2. As you can see, the DTD describes a hierarchical structure, and the XML file associated with the DTD must follow these definitions.
Since DTD is the simplest way to describe XML files, we will not go into detail on XSD or Relax NG, but for further reading on these two, we recommend W3C Schools [34] or Relax NG Homepage [29]. For further reading on DTD, you can visit W3C Schools' DTD tutorial [33].

9.4.4 XML parsing

To process xml files, you often have to use xml parsers. An Xml parser is an application or code, which extract the information from the data hierarchy.
There are many types of parsers, while the two most common ones are DOM, Document Object Model parsers, and pull parsers. The most common parser is the DOM, which can validate xml files with an XSD or DTD as explained in Section 9.4.3, and return an xml file as a structured tree. This is a quite cpu and memory intensive operation, and is not recommended used in applications designed for handheld units. An xml pull parser runs through the xml document once, one line at the time. As the parser runs through the document, data must be extracted and stored in variables. This means some disadvantages, but also some advantages. The most noticeable advantage with a pull parser is the speed and the memory usage. But as shown below, there are a few other differences between a DOM and a pull parser.

DOM Parser

- Advantages
  - Creates a tree model for later processing
  - Validation of documents with schemes
  - Better structure for large documents
  - Easy to alter or re-build xml documents
- Disadvantages
  - Uses a lot of memory
  - Heavy workload
  - A slow process

Pull Parser

- Advantages
  - Uses less memory
  - Easy on the CPU
  - Quick parsing/searching
- Disadvantages
  - Only one run-through, high demand on data organizing
  - No validation of xml documents
  - In most cases no way to generate/alter xml using xml-parser

We will focus on the pull parser, because this is best suited in mobile applications. For more information on DOM parser, please visit W3C Schools XML DOM Tutorial [35].

There are several open source pull parser available, all with their own strengths and weaknesses. The one that best meets our demands is the kXml parser. It is small, flexible and fast. It is also widely used, which means that documentation is available.
You can read more about the usage of kXml in PeerQuiz and PeerShare, in Chapter 14.1 and 20.2.

Part IV

Test application 1 - PeerQuiz

CHAPTER 10

Introduction

This part will describe the design and implementation of PeerQuiz. We will discuss requirements, challenges during the implementation and the testing of the application.

We chose to do develop a trivia game, because it describes a good scenario of spontaneous collaboration in a typical ad hoc situation. A trivia includes communication and interaction. It involves several participants, questions, answers and scores, and was designed to manage an infinite number of participants. The application is design as a hybrid-P2P model, and therefore requires one *masternode*. There can be many participants, *slavenodes*, however, it is the limitations of the masternode that determines the maximum number of slaves that can be connected at the same time. In most cases there is a maximum number of *seven nodes* simultaneously. This means the maximum number of participants can be eight. But since Bluetooth is constantly evolving, the PeerQuiz can theoretic involve 100 participants when Bluetooth can handle it.
PeerQuiz is designed and implemented upon the Peer2Me framework [22], which controls the actual Bluetooth communication between the nodes. This provides an abstractionlayer which helps PeerQuiz to easily establish communication without adding low-level code.
During the design and implementation of PeerQuiz, there have been to main focuses; utilize the Peer2Me framework in best possible manner, and create an intuitive and representable user interface. Since the PeerQuiz is a Java MIDlet design to run on a wide range of different handheld equipment, we encountered several limitations. More on this in Chapter 14.

CHAPTER 11

Requirements

This chapter describes a scenario, goals, functional requirements, use case description and quality requirements for the application PeerQuiz. An explanation for the different sections can be found in chapter 5.

11.1 Scenario

Joey agrees to meet his friend Christina at the local café to eat lunch. They are chatting and discussing various subjects, but is starting to get bored of each other. Suddenly Joey sees three of his friends entering the café. Since both Joey and Christina are frequent users of PeerQuiz, they ask Joeys friends to join in and participate in an informal quiz. Joeys friends have never heard about PeerQuiz, but all of them have mobile phones except Vikraam who has a PDA with WLAN-capabilities. Joey tells them to download the program from the web using a web browser. As they are downloading, Joey sets the rules of the game;12 questions with a time limit of 15 minutes. Joey then chooses four categories, with three random alternative based questions in each category. By now, Vikraam and Joey's two other friends are finished downloading and have installed the quiz. Joey starts the search for participants. Joey finds his friends on his phone and starts the quiz. Joeys phone automatically sends out the questions to the other participants. After 20 minutes, Christina, Vikraam and the others submit their answers to Joey. Joeys mobile phone the sums up the answers, and declares a winner. They all have a big laugh, and buys Vikraam, the winner, a beer.

11.2 Goals

1. The application must allow a user to set up a new quiz competition.

2. The application must support several contestants/clients.

3. A quiz must contain a set of rules.

4. The application must store questions in an organized manner.

11.3 Functional requirements

The functional requirements shown in Table 11.1 are set up for this application to meet the goals above.

FR1	The application must allow a new quiz to be created
FR2	The creator of the quiz must function as the master node
FR3	The application must allow several different set-ups / rules of a quiz
FR3.1	The creator must choose the number of questions
FR3.2	The creator can select a quick quiz which uses default set-up so that the quiz can start immediately.
FR3.3	The creator can choose which category the questions will be taken from
FR3.4	The application must select random questions
FR3.5	The creator must choose the contestants
FR3.6	The creator may choose to set a time limit for the quiz
FR4	The creator should be able to participate in the quiz
FR5	The application must declare a winner after all the answers are received
FR6	The questions and answers must be stored in a well organized xml-file
FR7	It must be possible to import new questions to the xml-file
FR8	The application should be able to export the xml-file to other devices
FR9	The participants must be able to quit at any time

Table 11.1: *Functional requirements for PeerQuiz*

11.4 Usecase description

The use-case diagram shown in Figure 11.1, illustrates how the functional requirements relate to each other and the user of the application.

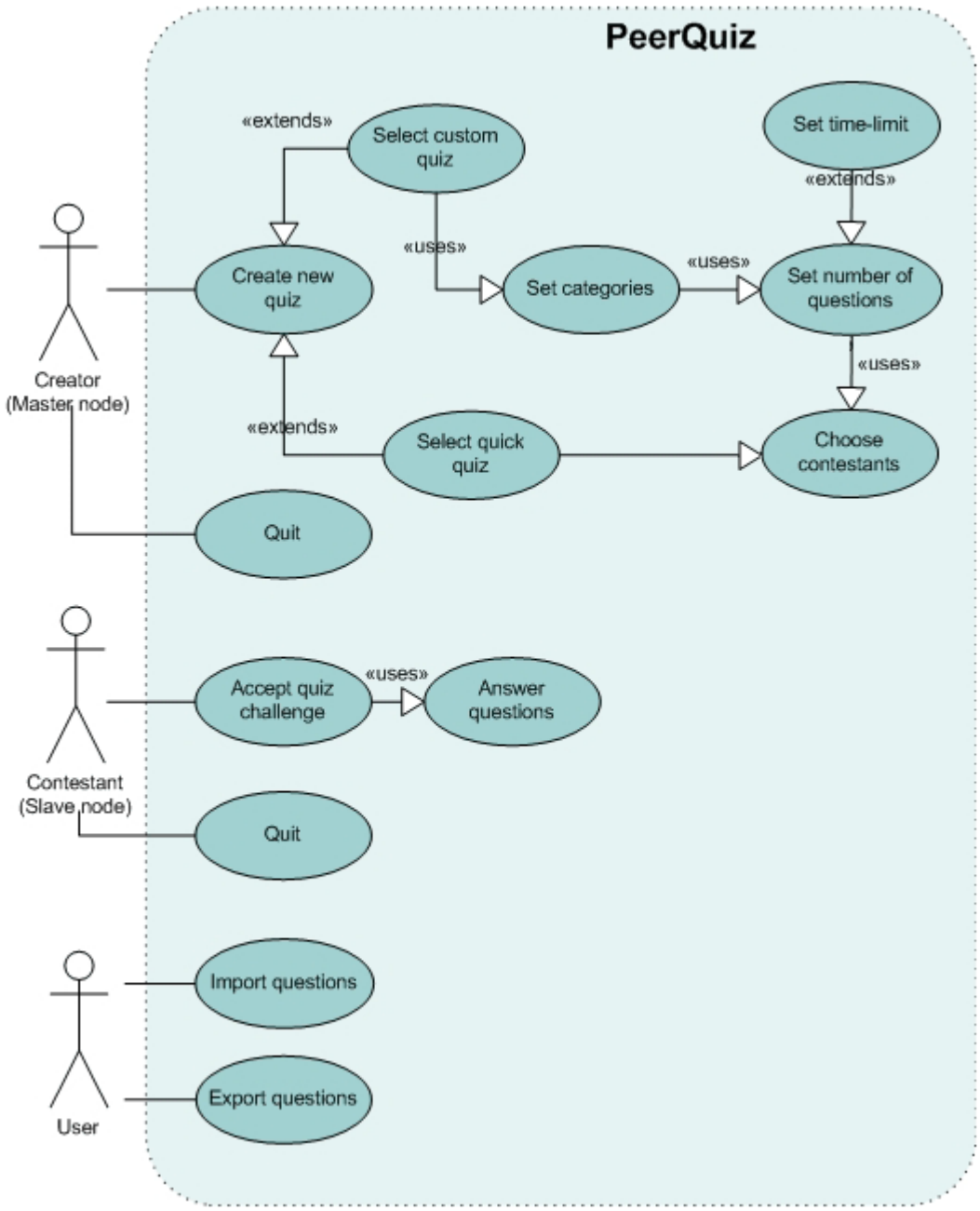


Figure 11.1: *Use case model for PeerQuiz*

11.5 Quality requirements

An important thing to keep in mind when designing applications, is to consider the non-functional aspects; The quality requirements. They are often just as important as the functional requirements.
This Section describes the quality requirements we find important for this application.

11.5.1 Usability

U1 – Inform the user of current process	
Source of stimulus	User
Stimulus	User interacts with PeerQuiz and the application starts doing something in the background that takes more than 1 s
Environment	Runtime
Artefact	PeerQuiz application
Response	The user should be informed of what is going on by displaying a progressbar or similar
Response Measure	The user should be informed immediately in all such cases

Usability requirement U1 is important so that the user can see that the program is still alive during time consuming operations.

11.5.2 Testability

T1 – Log file	
Source of stimulus	Developer
Stimulus	Developer wants to check if the application is doing what it is supposed to do by checking a log file.
Environment	Runtime
Artefact	PeerQuiz application
Response	Application logs all events, exceptions and state changes and stores them in an xml file.
Response Measure	Logging should be easily switched off

Since the display on a mobile unit is limited, it is hard to display possible errors in a sensible way, therefore the application creates a log-file containing information which can be accessed at all times.

11.5.3 Modifiability

M1 - Add new questions	
Source of stimulus	Developer
Stimulus	The xmlfile is extended with new questions. The developer should not need to change any code.
Environment	Design time
Artefact	Xml reader class
Response	There is no need to change the code.
Response Measure	Nothing to measure.

M2 - Add more alternatives to questions	
Source of stimulus	Developer
Stimulus	The questions in the xml file is extended with more alternatives, e.g. a question has now got five alternatives. The developer should not need to change any code.
Environment	Design time
Artefact	Xml reader class
Response	There is no need to change the code.
Response Measure	Nothing to measure.

M3 - Change structure of xml file	
Source of stimulus	Developer
Stimulus	The structure of the xml file is changed. The developer should only need to change code in the class which parses the xml.
Environment	Design time
Artefact	Xml reader class
Response	The change in code should only occur in the class that parses the xml.
Response Measure	The change should be accomplished within an hour and no ripple effects should happen.

The handling of questions in PeerQuiz involves extracting data from an xml-file. Since the xml-file is not a part of the actual codebase, it is very important that its content and structure can be altered without the need of larger modifications to the application.

11.5.4 Availability

A1 - Broken link	
Source of stimulus	Slave or master
Stimulus	A slave loose connection with the master
Environment	Runtime during quiz
Artefact	The slaves application
Response	The slave must store current status and the master must register that the connection was broken
Response Measure	Status must be saved 100% of the time and master must disconnect the slave after 5 attempts at pinging him/her.

As mentioned earlier, mobile communication is vulnerable to unforeseen loss of connection, and it is important to keep status of the gameplay. An availability tactic should be implemented in order to deal with the detection of lost connections.

CHAPTER 12

Dependencies

This chapter points out the applications dependencies in order to function properly. Even though Java is meant to be platform independent, there are a lot of different J2ME implementations on todays mobile phones.

12.1 Packages

This application uses the following packages:

- *J2ME library* - The standard J2ME library.
- *Peer2Me.jar* - The Peer2Me framework.
- *jsr082.jar* - The Java APIs for Bluetooth.
- *kxml2-min.jar* - An open source xml pull parser designed for use on devices with limited resources.

The Peer2Me framework uses the J2ME library and the jsr082.jar package. This application uses one additional package;kXml's kxml2_min.jar. This package can be run on most mobile phones.

12.2 Supported mobile phones

We are using the package jsr82 in order to access the Bluetooth device on the mobile phone. There are many units on the market that supports Bluetooth, but they must also support Java MIDP2.0. The newest Sony Ericsson mobile phones supports the use of jsr82, see Table 18.1. The phones that supports jsr82, also supports MIDP 2.0.

Manufacturer	Model	JSR 82 support	At our disposal
Sony Ericsson	W900i	Yes	No
Sony Ericsson	W800i	Yes	Yes
Sony Ericsson	W600	Yes	No
Sony Ericsson	W500	Yes	No
Sony Ericsson	K750i	Yes	Yes
Sony Ericsson	K600	Yes	No
Sony Ericsson	V600	Yes	No
Sony Ericsson	z520	Yes	No
Sony Ericsson	p900	Yes, but with bugs	Yes
Siemens	s65	Yes	Yes

Table 12.1: *Supported mobile phones*

CHAPTER 13

Design

The design of this application is centered around the *Quiz* class, which acts like the backbone of the application and extends the MIDlet-class and therefore inherits the ability to display forms and do action-events. *Quiz* also acts as a parent to the other pieces of the application. It holds methods to control the progress and program flow. These methods are called by many of the child classes to change program states and give the user different views.

The application is divided into to four main packages to minimize dependencies and to ensure the abstraction of code.

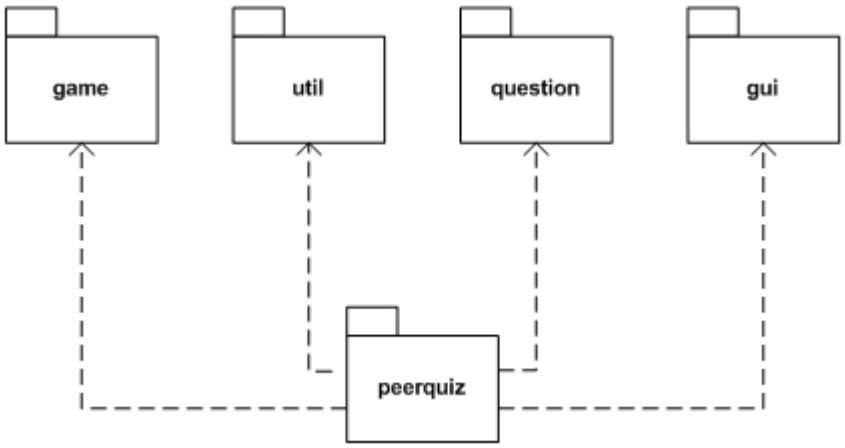


Figure 13.1: *Package diagram for PeerQuiz*

Each package has its own focus, and as seen in Figure 13.1 they are all on top of the peerquiz package(root).

13.1 The PeerQuiz Package

This is the root package, and contains only one class and one interface. This is the starting point of the application.
This package is illustrated in Figure 13.2.

Quiz: This class is the starter class and functions as a backbone in the application. The quiz-class creates instances of the different objects and displays the different gui-classes on the mobile units screen.

Constants: This interface contains several variables used in *quiz*.

13.2 The Game Package

This package contains the classes, which handles the communication and most of the dataflow. The game package consists of six classes. These classes handles the communication between the participants in the quiz.
This package is illustrated in Figure 13.3 and Figure 13.4.

QuizMaster: This class creates and handles the framework components required for the masternode to communicate with the slavenodes. It also contains the masternode's methods for creating and receiving messages.

QuizSlave: This class creates and handles the framework components required for the slavenode to communicate with the masternode. It also contains the slavesnodes' methods for creating and receiving messages.

QuizParticipant: A simple class for holding information about a participant i the quiz. Keeps track of name, address and score.

QuizParticipants: A class that extends *Vector* to inherit enumeration functionality. Meant to contain several Participant-instances and has implemented methods to easily add and remove participants without having to cast the data.

QuizRule: Holds the information about a quiz' rules; time limit, number of questions and categories.

QuizMessage: A wrapper-class for the Peer2Me framework's `Message`, adds extra functionality that makes it easy to send questions and other messages. Since `QuizMessage` extends `Message`, it's not necessary to alter the framework to send a message, you can simply send the `QuizMessage`.

13.3 The Gui Package

This package contains the classes which extends a `Displayable` object, most of them a `Form`, which can be displayed to the user, hereafter referred to as a *screen*. This package handles all the interaction with the user.
This package is illustrated in Figure 13.2 and Figure 13.6.

SplashScreen: This screen shows the user a logoimage and a continue-button.

MainMenuScreen: This screen shows the user three buttons, each with an action-event. The user must here choose weather to be a masternode or a slavenode. Or enter the settingsscreen.

SettingsScreen: The only choice available here is to active the log-button, which enables the user to view a log any time during the quiz to detect problems or control the game flow.

SlaveScreen: The screen where the participant enters his or hers name a enters a listening mode and enables the node to be discovered by the masternode.

MasterChoiceScreen: In this screen the user is first presented with two choices, quick-quiz or custom quiz. If he/she chooses quick-quiz, there will be a standard quiz with predefined a *QuizRule*. If custom quiz is chosen, the user will be presented with options to choose a different time limit, a different number of questions or choose a single category instead of all categories.

MasterScreen: This is where the user enters his or hers name and search for slavenodes. Will display information about the nodes found.

CommunicationScreen: This is a common class for both master and slave, and behaves in some way similar to both parts. The slave node is showing the rules of the game, and the progress of receiving questions, while the masternode shows the progress of sending the questions.

QuestionScreen: The screen where the questions are displayed. Contains simple actionevents for displaying the next question in line, and submitting when finished. This will be the last screen shown to all users.

BarItem: This item shows the user a non-static progressbar. Contains its own thread and images.

13.4 The Question Package

This package contains classes which represents the question-objects. These are classes which builds up a hierarchy of objects to identify a `QuestionCollection` with `Question`-objects, that contains several `QuestionAlternative`-objects. `QuestionCategory` is used to identify a question when in `QuizRule`.
This package is illustrated in Figure 13.7.

QuestionCollection: Extends `Vector` to inherit enumeration functionality. Meant to contain several `Question`-instances and has implemented methods to easily add and remove questions without having to cast the data. has also got methods for getting next or previous question in line.

Question: A simple class which holds the main entities in a question; the questiontext, the answer, the weighting of the question and the alternatives the user can choose to answer.

QuestionAlternative: A class to provide a simple object to imitate a hash-table with a String-index.

QuestionCategory: Contains only a String, but is used to objectify the design.

13.5 The Util Package

Contains several classes used to provide functionality used by the quiz; a log function, utilizing the *Record Store*, the implementation of an XML-parser and a simple timer.
This package is illustrated in Figure 13.8.

QuizLog: Provides the ability to temporary store and retrieve `QuizLogMessage`-objects to/from memory, and therefore makes it possible to track events in the quiz and report exceptions. Extends Vector to inherit enumeration functionality, but with simple methods to add and remove messages and sort output by severity.

QuizLogMessage: Stores the information on every log-entry by text, timestamp and severity.

QuizRecordStore: Utilizes the *Java Record Store* and the *Java Record Store Enumerator* to provide simple functionality for creating, updating and deleting records from the mobile unit's recordstore.

XMLTool: Uses the open source xml pull parser kXml to traverse an XML-document, in this case to find questions. Provides public methods for finding questions and categories.

QuizTimer: Provides a simple timer which controls the ping-function and the time limit in the quiz. Extends the *java.util.TimerTask* which runs as an own thread.

13.6 Class Diagrams

This Section contains the class diagrams for all the packages in PeerQuiz.

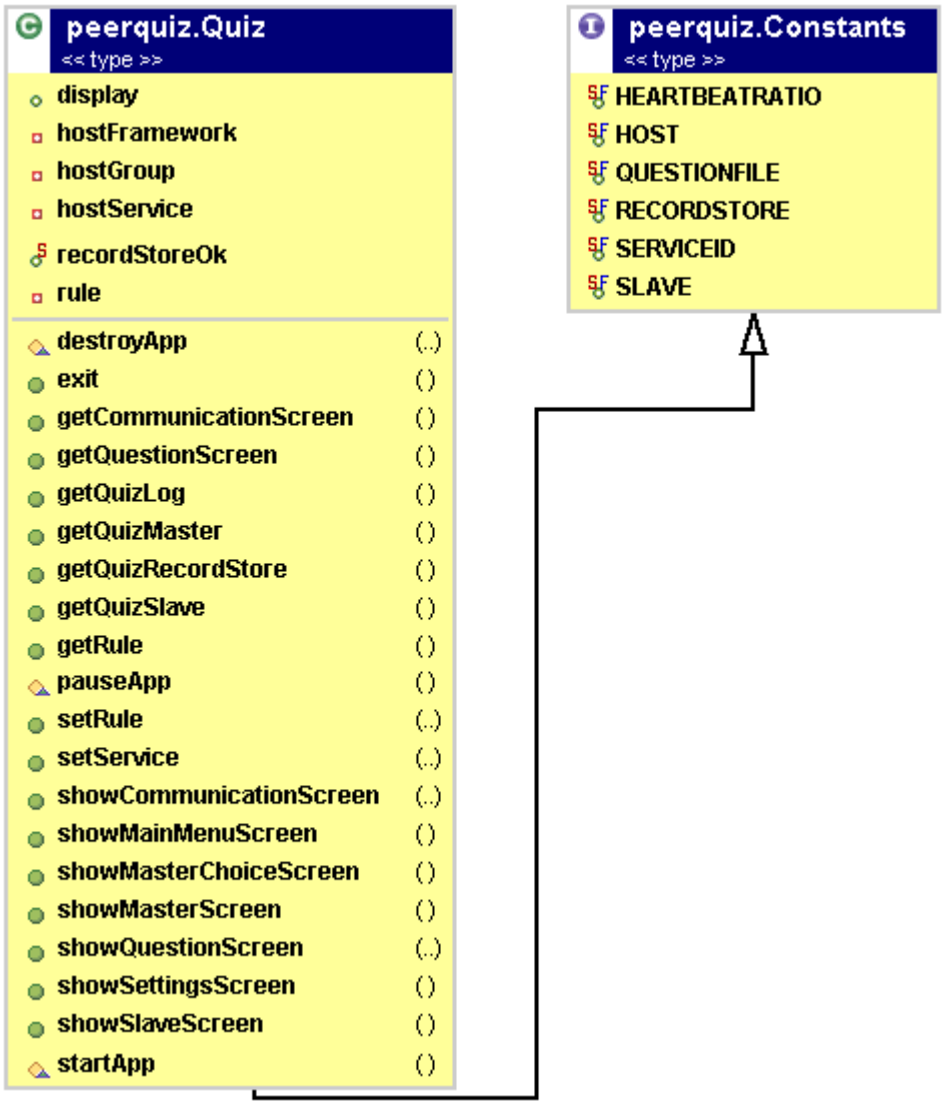


Figure 13.2: *The peerquiz package*

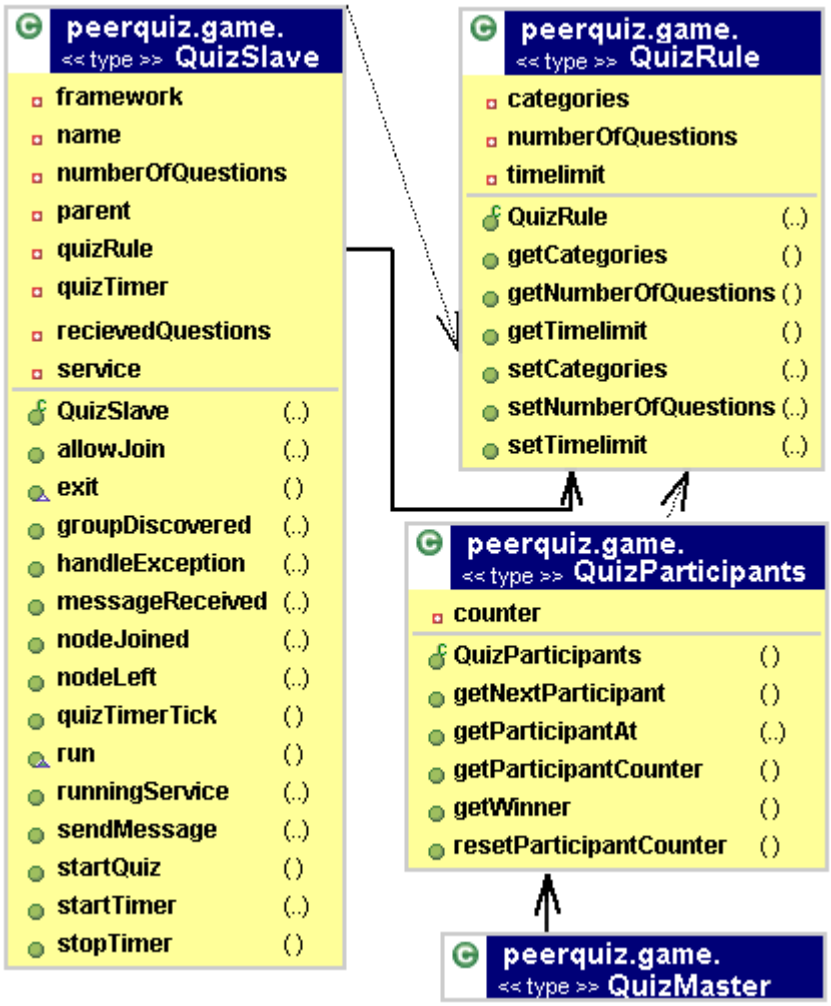


Figure 13.3: *The game package*

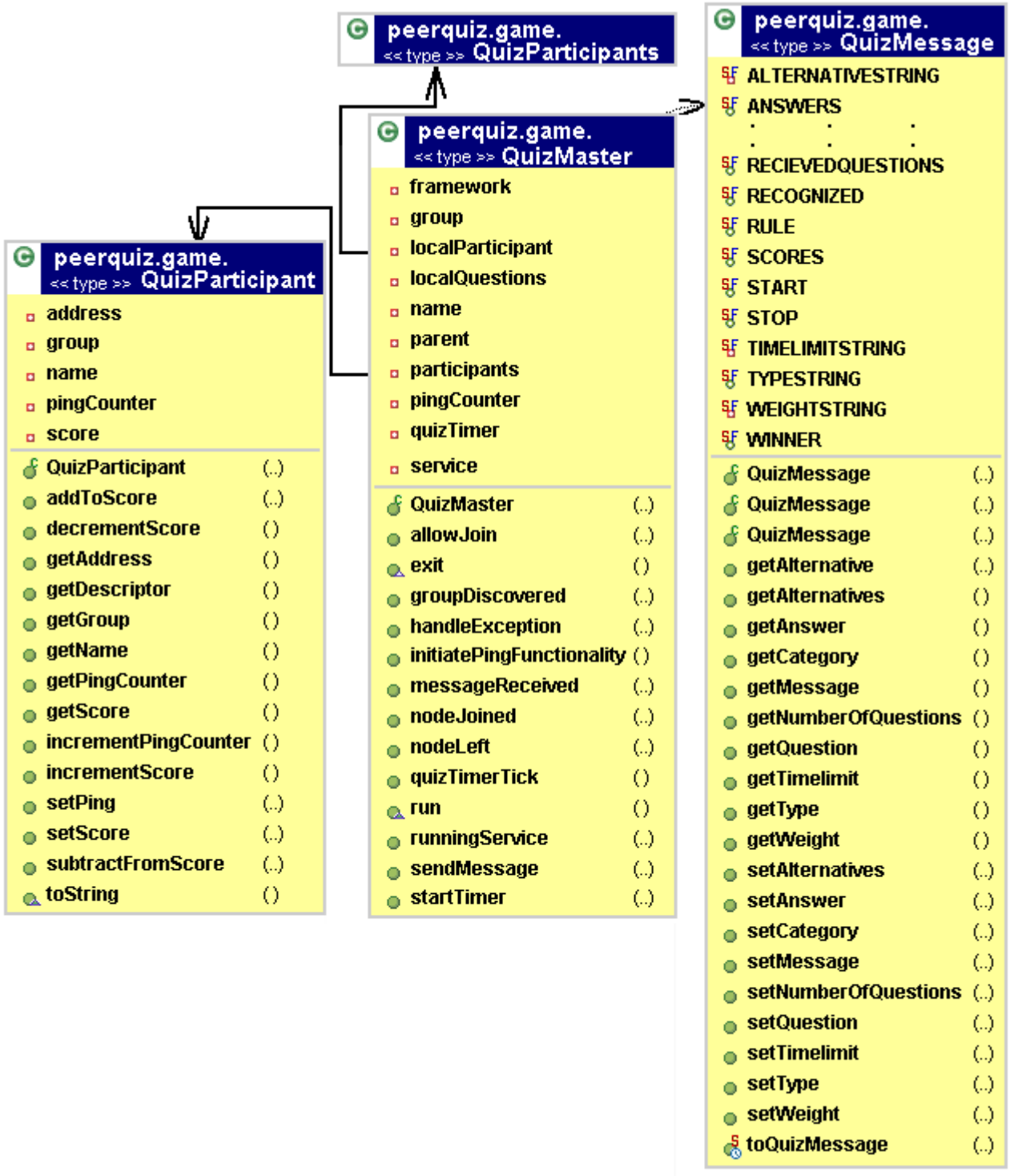


Figure 13.4: *The game package part 2*

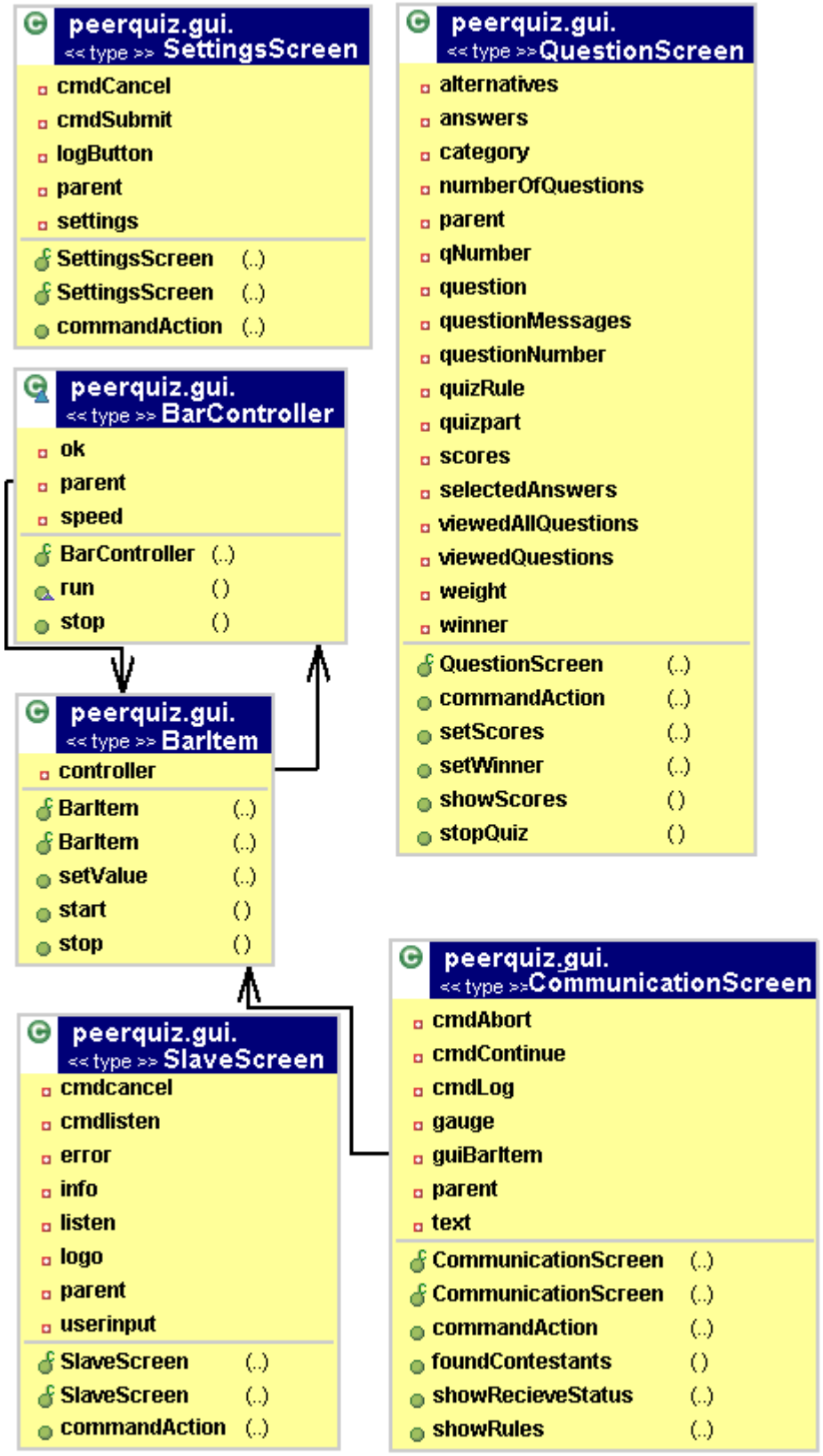


Figure 13.5: *The gui package part 1*

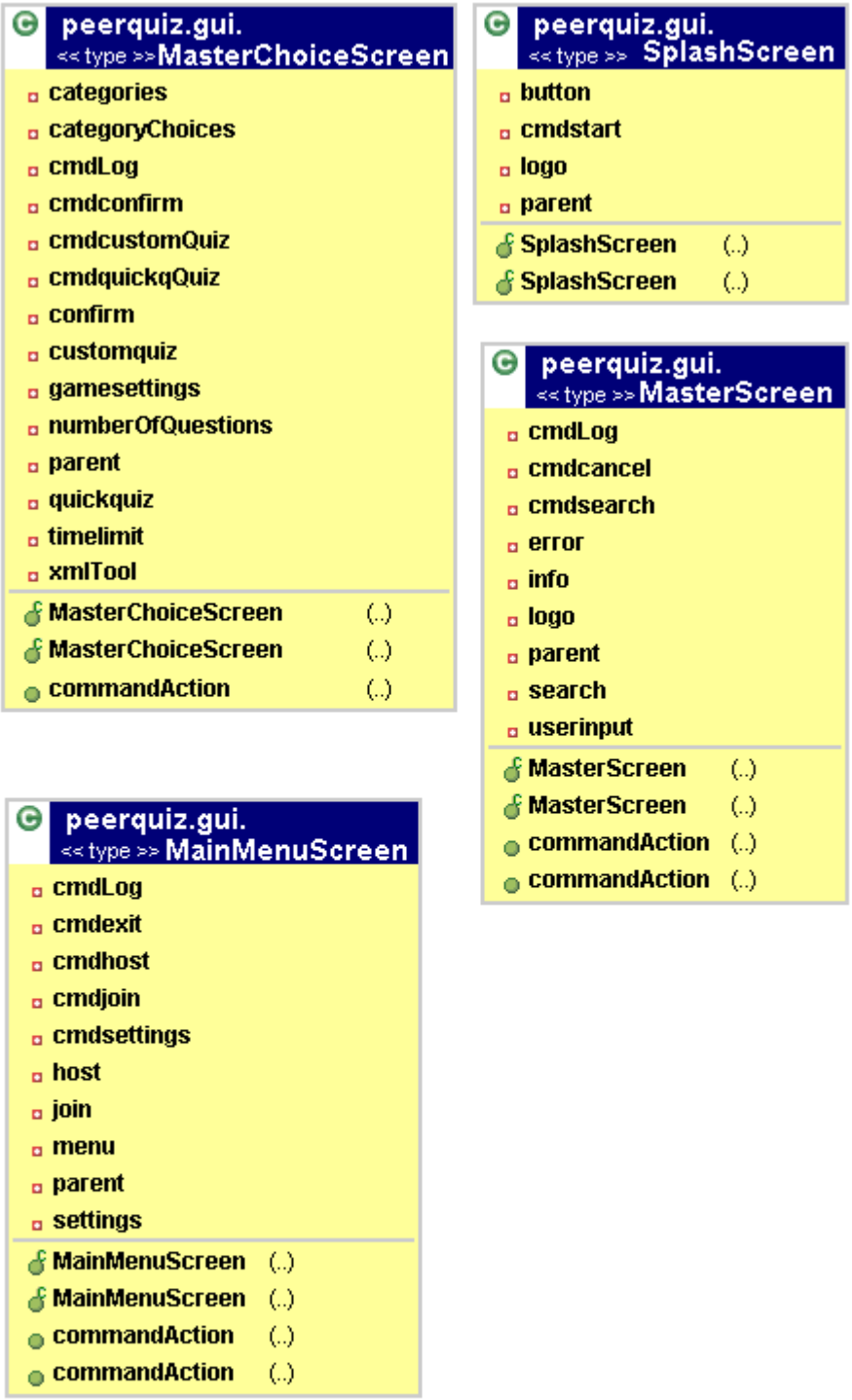


Figure 13.6: *The gui package part 2*

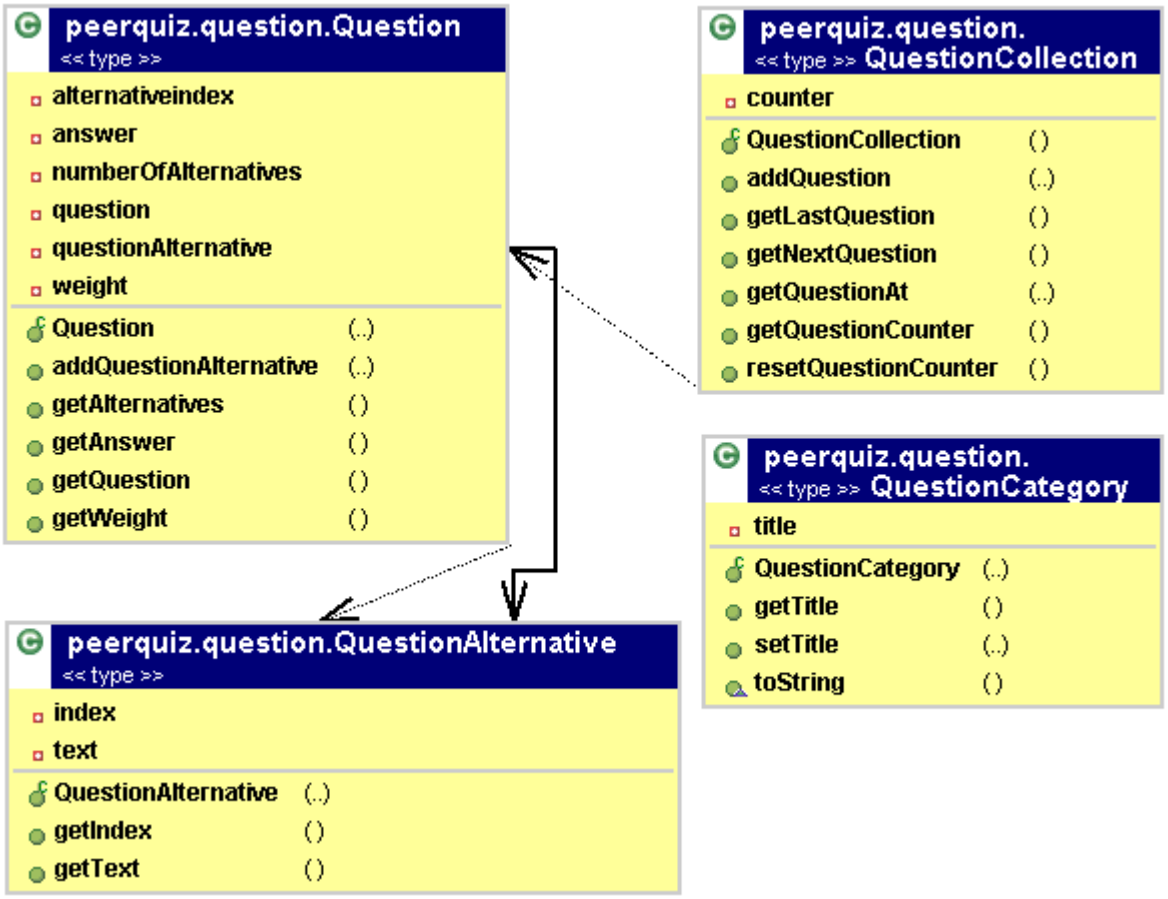


Figure 13.7: *The question package*

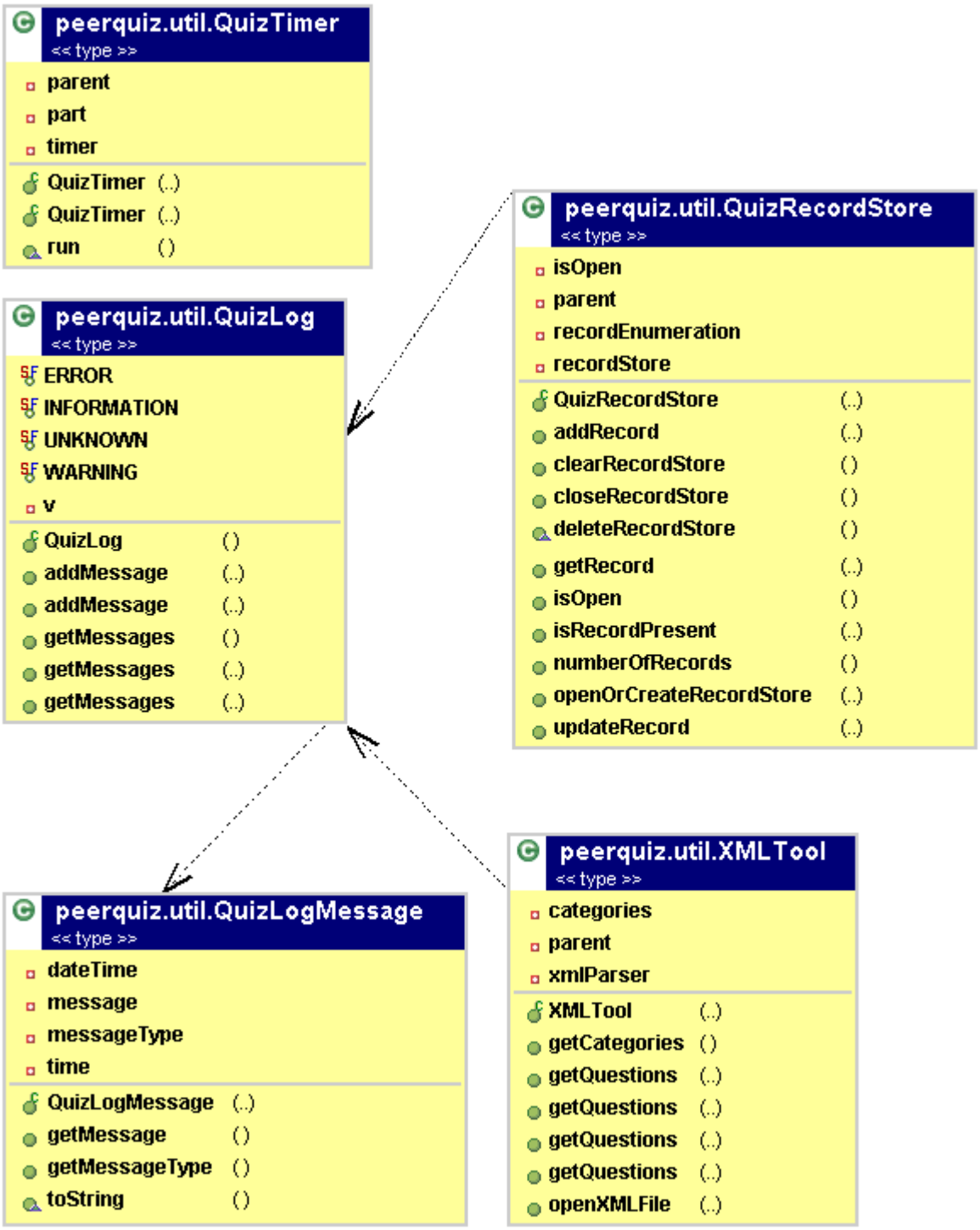


Figure 13.8: *The util package*

13.7 Program flow

This Section will describe the flow in PeerQuiz using two diagrams; a Gane Sarson Flow Diagram, Figure 13.9 and an UML Activity Diagram, Figure 13.10.

Gane Sarson Flow Diagram

The Gane Sarson Flow Diagram shows in which order the graphical user screens appear. This gives an overview on how the application will appear to the user. It is important to keep in mind that this diagram does not involve any action events.
The splashscreen is the first screen shown to the user. This is followed by a screen showing a menu giving the user three choices. The path is now different depending on whether the user acts as a master or a slavenode. Communicationscreen and Questionscreen is common for both parts.

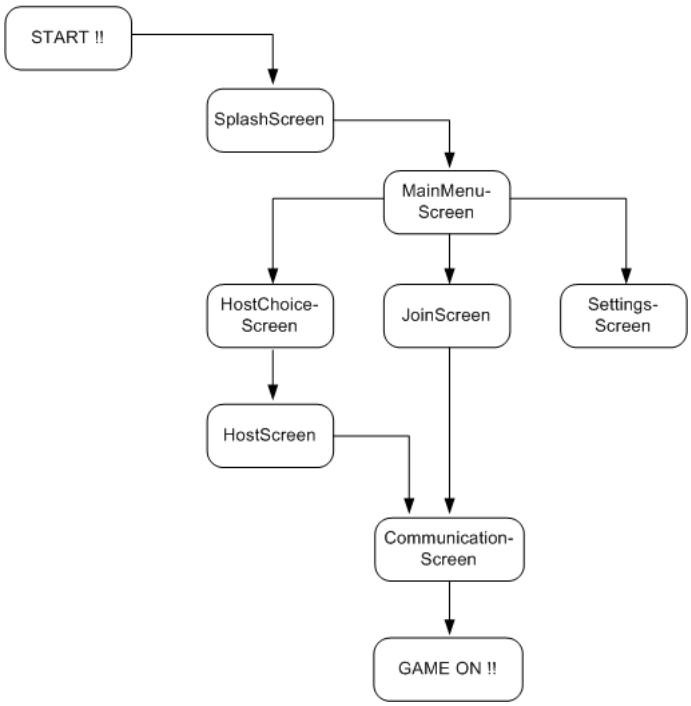


Figure 13.9: *PeerQuiz flowchart*

UML Activity Diagram

The UML Activity Diagram describes in which stages the user interacts with the application. It describes the choices the user might meet, and the consequence of making each choice. In the end, after the final state, the application exits.

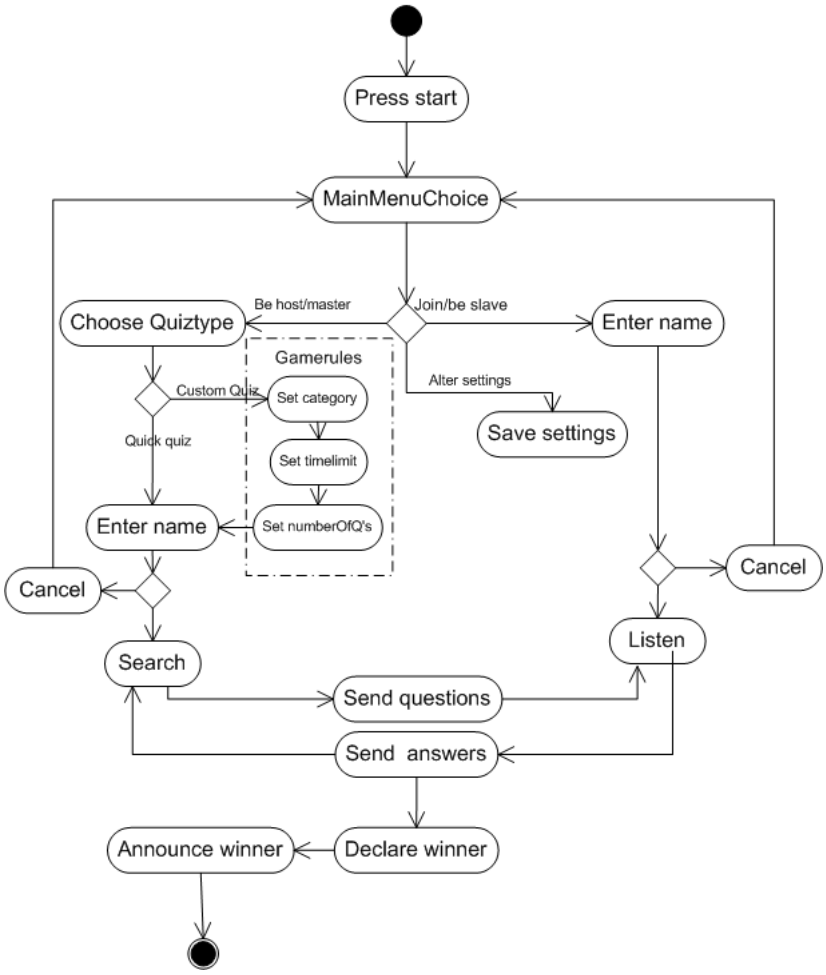


Figure 13.10: *Peer Quiz UML activity diagram*

CHAPTER 14

Implementation

This Chapter describes the implementation details of the PeerQuiz applications, and some code snippets from important features. We have chosen to focus on the main parts of the application; *Xmlparsing*, *ping/echo tactic*, *graphical user interface* and *communication*. Xmlparsing is an important feature in the application, since it is the way the questions are organized, and the parser is needed to extract the data from the xmlfile.
We have also met the availability requirements by implementing an availability-tactic in PeerQuiz. This is described in detail in Section 14.2. This section will also discuss the GUI-components used, and the details of communication.
The complete source code for this application can be found on the attached CD.

14.1 Xml parsing and Questions.xml

This Section will describe the xml format of the file which stores the questions, and how PeerQuiz utilizes this file.
In PeerQuiz, the questions are stored in an xml file to ensure modifiability. As explained in Section 9.4, xml stores the information hierarchical in nodes, where each node can contain subnodes, and each node contains several attributes. This is a excellent way to objectify data and make sure the data is consistent and expandable. Since PeerQuiz has no limitation on the number of questions, the data structure of the questions does indeed need to be expandable.

In todays version of the Peer2Me framework, you can only send messages with ASCII characters. See Section 21.1.2 in Sars & Norums master thesis [22] for more explanation. This gives us two ways of utilizing xml in PeerQuiz. We can either convert the xml document(or parts of it) to a long string, and then send the string. Or we can extract data from the xml document and create several messages or messageparts. In PeerQuiz we are using the second alternative, while we in PeerShare are using the first alternative.

Listing 14.1: *Xml-format, from* `Questions.xml`

```
1 <Quiz>
2     <Category name="Geography">
3         <Question text="Where is Timbuktu?" answer="2" weight="5">
4             <Alternative id="0" text="Peru"/>
5             <Alternative id="1" text="Tunisia"/>
```

```
            <Alternative id="2" text="Mali"/>
        </Question>
    </Category>
</Quiz>
```

As we can see, the top node is quiz, and there is a categorynode, with a questionnode inside. The questionnode carries several alternatives. When you have this structure of data, it is quite simple to traverse the xml and fetch the nodes which match your preference. To validate and explain this xml document, you can, as explained in Section 9.4.3, use a DTD. The DTD for *Questions.xml* are shown in Listing 14.1 below.

Listing 14.2: *DTD for* `Questions.xml`

```
<!ELEMENT Alternative EMPTY>
<!ATTLIST Alternative
 id CDATA #REQUIRED
 text CDATA #REQUIRED
>
<!ELEMENT Category (Question)>
<!ATTLIST Category name CDATA #REQUIRED>
<!ELEMENT Question (Alternative+)>
<!ATTLIST Question
 text CDATA #REQUIRED
 answer CDATA #REQUIRED
 weight CDATA #REQUIRED
>
<!ELEMENT Quiz (Category)>
```

14.1.1 kXml Pull Parsing

To achieve parsing the xml document in a simple manner, we have chosen to implement the kXml parser, see [20]. As stated in Section 9.4.4 earlier in this document, kXml is a pull parser designed for mobile applications with a rather small footprint. This is essential when keeping in mind that the Peer2Me framework must also be included in the application.

14.1.2 Put to use

The parser is using `xmlParser.nextToken()` to traverse line by line in the XML-document and then returns the node tagtype (starttag,endtag and so on). When you found your tagtype, in other words the nodetype you are looking for, you can easily check the identity by calling `xmlParser.getName()`. To get the content of a node, simply use `xmlParser.getAttributeValue()` with the proper attribute-id.
As mentioned the Peer2Me framework has not yet got support for sending xml in other data types than wrapped as a string, but this could be a possible expansion to the framework in the future.

14.2 Availability tactics and ping-functionality

This Section addresses the availability requirement A1 in Section 11.5.4, and how it is solved in PeerQuiz.

Mobile Bluetooth communication have a hard time when it comes to availability. As soon as two devices get out of range of each other, there is no immediate reaction even though they are not connected anymore. You might think the Peer2Me framework got this covered, but there are no functionality in the framework to take care of this incident, and you will simply get an exception or an error telling you that the message could not be sent or the node couldn't be reached.
Utilizing the framework's messagesending, PeerQuiz has got the ping/echo-functionality implemented to ensure availability. This is done by bouncing a message from the masternode to the slavenode and back again.

14.2.1 Ping/echo

The ping/echo-tactic is quite similar to the heartbeat-tactic, but instead of the slave sending a "I'm here"-message every now and then, the ping/echo-tactic consists of making the masternode request the message. Using a timer, *QuizTimer*(see Section 14.2.2), the PeerQuiz is at a given interval sending a request to all connected slaves telling them to respond. If a slavenode doesn't respond on any of the five last requests, the node is considered lost and removed from quiz.
This functionality will provide awareness to the masternode whether the other participants are connected and within reach or not. This will increase the amount of data transmitted from the masternode, but there are no other well defined methods for achieving this in ad hoc network communication. The size of the message sent back and forth are very small, and should not make any difference if the number of participants is less than seven.

14.2.2 QuizTimer

To ensure that the ping-request is sent to the slavenodes regularly at a specified interval, PeerQuiz uses its own *QuizTimer*. QuizTimer is a class which extends *java.util.TimerTask* to make sure it runs as an own thread, and uses *java.util.Timer* to schedule happenings at certain intervals. The QuizTimer is then configured to call on a method on the masternode when the desired time is reached. This is referred to as a *tick*. The timer can be scheduled in two ways, either to produce a single tick after a given delay, or produce ticks at given intervals. The last mentioned is used in ping/echo-functionality, while the single-tick-feature is used when the quiz has a time limit.
This timer can easily be implemented into the Peer2Me framework to provide this availability fault-detection tactic. For more information about ping/echo-tactics, please see "Software Architecture in Practice" by Bass, Clements and Kazman [5].

14.3 GUI - Graphical User Interface

The graphical user interface is very important to the application's usability. You might have the best program code and algorithms available, but without a sensible user interface to ensure intuitive interaction with the user, your application is less user friendly. Most application's popularity depends on a good user interface.
In J2ME, creating a good user interface is very difficult due to a limited selection of gui components. You have two main choices; you can either use a form and then add simple components as text fields and images. Or you can use a canvas and then draw strings and images at an absolute position. This will create a better user interface, and as mentioned earlier, enable using the numpad. The drawback is that it's not very flexible nor easily created. You also have to keep in mind that the components might appear differently depending on the handheld unit running the application.

14.3.1 Form

In PeerQuiz forms are used, and in the gui-package (see Section 13.3) most of the classes represents a single form which can be displayed on the screen. For example the `QuestionScreen` extends `Form` and can be displayed very easily in the MIDlets main class. See Listing 14.3.1 to see the codesnippet which performs a "screensetting"

Listing 14.3: *Setting new screen codeexample*

```
1 //Getting the MIDlets displayhandler
2 Display display = Display.getDisplay(this);
3
4 public void showQuestionScreen()
5 {
6  //Creates the screen object that extends Form
7  questionScreen = new QuestionScreen();
```

```
8
9   //Puts the screen "on display"
10  display.setCurrent(questionScreen);
11 }
```

As earlier mentioned, the Quiz-class is the backbone of the application, and exposed many methods for changing screens. This makes it easy to add new screen by simple creating a class that extends Form and add a new set-method in Quiz.
As a main theme, we are using mostly the *javax.microedition.lcdui.ImageItem* with an associated command as buttons. This is the most common way of creating buttons in MIDlets. Where the space on the display is required to showing text, the buttons are as actions on the mobile units physical buttons.
To always show the user what is happening, we are using the standard progressbar in J2ME; *javax.microedition.lcdui.Gauge* This is a normal progressbar showing a graded bar, with a percentage on some handheld units, for example the Sony Ericsson W800i [2]. Since we felt that *gauge* didn't give sufficient feedback to the user, with created our own progressbar, *BarItem*, which is covered in Section 14.3.2

14.3.2 The PeerQuiz progressbar, BarItem

The *BarItem* is a displayable item used to let the user know that something is happening; give the user feedback.
The BarItem extends ImageItem, but utilizes its own class, *BarController* which runs as an own thread, constantly changing the BarItem's displayed picture. One might call this a slideshow consisting of five images, changing at a specified interval. The BarItem is shown in three steps in Figure 14.1.

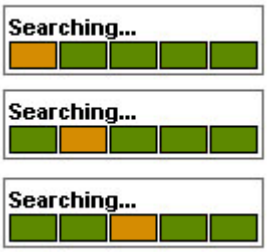


Figure 14.1: *BarItem graphics*

14.4 Communication

The primary goal with PeerQuiz is to utilize the Peer2Me-framework in a P2P setting; make several handheld units communicate over Bluetooth.
The framework consists of several components which must be used to communicate.

14.4.1 Initialization

The first operation that needs to be done, is initializing the framework-component. As shown in Listing 14.4.1, only two lines are actually necessary to initialize the framework.

Listing 14.4: *Initializing the framework*

```
106   //Initializes the framework and setting listeners
107   framework = Framework.getInstance(
108     "HostCommunicator",
```

```
109       "Quiz Master",
110       "no.ntnu.idi.mowahs.project.bluetooth.network.BluetoothNetwork");
111     framework.init();
```

The next thing to do is to determine the functionality you need, and then add the correct handlers. This depends whether you need to search for slave nodes, use messagesending/receiving and so on. The three handlers needed in PeerQuiz is shown in Listing 14.4.1.

Listing 14.5: *Setting listeners and handlers*

```
112     framework.setGroupDiscoveryListener(this);
113     framework.setMessageSubscriber(this);
114     framework.setExceptionHandler(this);
```

These handlers require their own methodstubs in the same class, since the framework implements an interface, and the class which uses the framework must also implement several interfaces. For more information, see the master thesis by Norum and Sars [22].
The next step is to register the framework-components with each other. This is to ensure interaction between the components. There are three main components that are needed to utilize most of the functionality; *Framework*,*Service* and *Group*. In PeerQuiz this is done by registering the group in the service, then register the service in the framework. This is shown in Listing 14.4.1 shown below.

Listing 14.6: *Registering components*

```
121     //Registrering the group in the service
122     service.setGroup(group);
123
124     //Registering the service in the framework
125     framework.registerService(service);
```

14.4.2 Master vs. Slave

Since the Peer2Me framework distinguishes between the masternode and the slavenodes, and since there can be only *one* masternode, this must come in consideration when designing and implementing the applications. In PeerQuiz, there is a masternodeclass, *QuizMaster.java* and a slavenodeclass, *QuizSlave.java*. They share some of the same functionality, but are quite different i how the work. We created therefore two separate classes.
While the slavenode automatically goes into a listening mode after registering the components, the masternode must actively search for slavenodes. This is in PeerQuiz done easily by calling one single method as shown in Listing 14.4.2.

Listing 14.7: *Searching for slavenodes*

```
137  private void searchForNodes()
138  {
139   this.parent.getQuizLog().addMessage(QuizLog.INFORMATION,"Searching for participants");
140   framework.startGroupSearch(service);
141  }
```

The framework requires that the class that uses the framework, exposes a set of predefined class-stubs. These methods are triggered by the framework e.g. when a message is received or a slavenode has joined the group. This is done by Sars and Norum to ensure that the framework is abstract enough to be implemented easily.
An important thing to notice, is that in the slavenodes method `groupDiscovered(..)`, the slave will register a groupmonitor and the masters group to its own framework instance. Listing 14.4.2 shows the code.

Listing 14.8: *Slavenode is discovered by masternode*

```
242  public void groupDiscovered(Group group)
243  {
244   //Sets the groupmonitor
245   group.setMonitor(this);
246
247   //Join the group found
248   framework.joinGroup(group);
249  }
```

14.4.3 Sending messages

The key feature of the communication between the units connected to each other is messagesending. The ability to transmit data to one or more nodes. This is done by using the framework-components to send a QuizMessage-object, as explained in Section 14.4.3. As mentioned in Section 14.1, PeerQuiz is extracting data from the xml file and creating Question-objects server side. This is handled by `QuizMaster` and `QuizMessage`.

QuizMessage

QuizMessage is a class that functions as an encoder and a decoder for the messages sent in PeerQuiz during a game.
QuizMessage extends *no.ntnu.idi.mowahs.project.domain.Message* to inherit the original functionality provided by the Peer2Me framework. Since this is provided, it gives the opportunity to send a QuizMessage without have to cast it to a Message-object. As shown in Figure 13.4, QuizMessage defines multiple constructors which can create a new instance of QuizMessage, all depending on what type of message you want to send. The main advantages by doing it this way, is that you don't have to create many textmessageparts every time you e.g. send a message, QuizMessage does it for you. QuizMessage stores the input from the appropriate constructor and adds the correct textmessageparts an internal messageobject. The QuizMessage is now ready to be sent in the framework the same way as the original message-object is sent, by creating the QuizMessage and adding recipients, and then calling `framework.sendMessage(questionMessage,service)`.
When the message is sent and the other joined nodes receives it, it is received as a `Message`, not a `QuestionMessage`. This is because of the framework. To avoid having to extract the data from the message using many message.getMessagePart().getFieldValue() calls, the toQuizMessage(Message) method in QuizMessage should be called, and it creates a new QuizMessage object based on the received Messageobject. This is done by analyzing the received message's messagetype and calling the correct constructor in QuizMessage. Now that you have the QuizMessage instance, several methods are provided for easy access to the data that was sent.

14.5 Code statistics

In order to illustrate the size of the application, we have in Table 14.1 included code statistics for the PeerQuiz application.

Aspect	**LOC**	**# of classes**	**Avg. # of methods**	**Avg lines per method**
peerquiz.gui	966	10	5.6	15.06
peerquiz.game	942	6	15.33	5.88
peerquiz.util	674	5	6.8	10,13
peerquiz.question	132	4	5	2.25
peerquiz	222	1	25	3.96
Total	2936	26	8.73	7.76

Table 14.1: *PeerQuiz code statistics*

CHAPTER 15

Testing

This chapter presents the covered functional and non-functional requirements described in Chapter 11.

15.1 Test of functional requirements

The application is tested in two different environments:

Sun's Wireless Toolkit 2.2 with included emulator running on an x86 computer using Windows XP SP2 and Java version 1.5.0_04.

Mobile phones on test subjects. Sony Ericsson W800i and Sony Ericsson K750i.

Table 15.1 lists the functional requirements set out in Chapter 11 and if the requirement is supported by the application. Two new requirements, FR10 and FR11 are also added.

Requirement	Text	New requirement	Result
FR1	The application must allow a new quiz to be created	No	Pass
FR2	The creator of the quiz must function as the master node	No	Pass
FR3	The application must allow several different set-ups / rules of a quiz	No	Pass
FR3.1	The creator must choose the number of questions	No	Pass
FR3.2	The creator can select a quick quiz which uses default set-up so that the quiz can start	No	Pass
FR3.3	The creator can choose which category the questions will be taken from	No	Pass
FR3.4	The application must select random questions	No	Pass
FR3.5	The creator must choose the contestants	No	Pass
FR3.6	The creator may choose to set a time limit for the quiz	No	Pass
FR4	The creator should be able to participate in the quiz	No	Pass
FR5	The application must declare a winner after all the answers are received	No	Pass
FR6	The questions and answers must be stored in a well organized xml-file	No	Pass
FR7	It must be possible to import new questions to the xml-file	No	Not implemented
FR8	The application should be able to export the xml-file to other devices	No	Not implemented
FR9	The participants must be able to quit at any time	No	Pass
FR10	The application should store data such as the nickname used in the last session	Yes	Pass
FR11	The application should automatically load saved nickname when the data is required	Yes	Pass

Table 15.1: *Fulfillment of functional requirements for PeerQuiz*

15.1.1 Comments on the implementation of the requirements

All of the functional requirements are implemented except a few which is regarding the xml-file holding the questions. Since the focus should be on utilizing the Peer2Me framework, manipulating of the xml-file is given lower priority.

FR3.2 The quick quiz settings are at this point sat directly in the code, but it is easy to utilize the java recordstore as it is done with the nickname.

FR3.5 Due to the restrictions or lack of functionality in the Peer2Me framework, the master is not able to choose which contestants that can join the game. They join automatically since they have the same serviceID as the master.

FR3.5 At this point only one winner is declared, even if several participants got the same score. The one who answered first, is the winner.

FR7 Import and merging of questions in the xml-file is not implemented because it has no influence or correlation with utilizing the Peer2Me framework.

FR8 As in FR7, export is not implemented either.

15.2 Test of quality requirements

U1 - Inform the user of current process - Implemented The user is at all times informed about what is going on, either with text or with a progressbar, the BarItem. All information is displayed on the screen, but an alternative could be to use some units integrated vibrator. The BarItem is moving graphical item applied to show the user that the application is alive.

T1 - Log file - Implemented The programmer has to manually specify which messages should be stored in the log. The log can be easily toggled in runtime on the mobile phone, but is not stored when the application exits. In order to store the file, the developer can use the *peershare.util.FileHandler* class and write the file to any location he wants. We did not choose to do so as viewing the log during runtime proved to satisfy our needs.

M1 - Add new questions - Implemented If or when the user decides to add extra questions, he or she does not need to change any code, just follow the standards of the xml-file to make sure the parsing is correct. As discussed in Section 9.4, kXml, the pull-parser used in PeerQuiz, does not offer the ability to verify xml-documents before they are parsed. it is therefore critical that the syntax of the xml-file is correct.
The number of questions that can be added is restricted by the hardware.

M2 - Add more alternatives to questions - Implemented PeerQuiz and the xml-parsing supports an infinite number of alternatives, and as for requirement M1, the syntax of the xml-file needs to be correct for the application to function properly.

M3 - Change structure of xml file - Implemented If the user decides to alter the structure of the xml-file which holds the questions, he or she only needs to alter code in
peerquiz.util.XMLTool.java. This file contains several methods to extract categories and questions from the xml-file, and it is in these methods the code needs to be redesigned. These changes does not take more than one hour to complete provided that the developer has got basic java and xml knowledge.

A1 - Broken link - Partially implemented This requirement is implemented fully on the masternode, but only partially on the slavenode(s). The masternode is using the ping/echo functionality described in Section 14.2.1 in Chapter 14. This means that the masternode is fully aware of the slavenodes statuses, but at this point the slavenode is unaware that he or she is out of range. As a solution to this, the ping/echo could be replaced by a heartbeat mechanism, making the masternode listen for nodes sending a heartbeatmessage, and then time out the nodes that does not give a life sign in a given period of time. This will give more uncertainty and are not as precise as a two-way communication e.g. ping/echo.

Part V

Test application 2 - PeerShare

CHAPTER 16

Introduction

This chapter describes the second test application, PeerShare. PeerShare is an application that can be used to share files between several mobile phones simultaneously. It is of course possible to send files on your mobile phone to other devices using the "send as Bluetooth" function implemented on most devices, but this is not good enough for a filesharing network. This application allows users to browse each others file before downloading.

First of all, a scenario is presented along with the goals of the application which the requirements are based on. The requirements for the application are then presented, both functional and non-functional. The application has several dependencies and only runs on some mobile phones. A selection of supported mobile phones are listed. The PeerShare application consists of several packages with several classes. In order for the reader to get an overview, the different packages and their contents are illustrated in a couple of class diagrams. Reasoning for the package structure and essential implementation details of the application is then introduced. How we used the Peer2Me framework is also included in that Section.

When we first created the requirements, we were uncertain if all of them were implementable, especially since our goal was to use the Peer2Me framework for all communication purposes. Testing the application is a vital part of all application development and the process of testing involves checking if all requirements are met. At the end of this part, we have included some documentation around the testing process.

CHAPTER 17

Requirements

This chapter describes a scenario, goals, functional requirements, use case description and quality requirements for the application PeerShare. An explanation for the different sections can be found in Chapter 5.

17.1 Scenario

Vikraam, Christina and Joey are three friends who all own a mp3 mobile phone. They are regular users of the application PeerShare and use their mobile phone as an mp3 player in addition to ordinary phone usage. They meet at lunch and start discussing their mutual passion music. They all launch PeerShare and starts browsing through each others music library at the same time. Vikraam sees that Christina has a sample of the brand new hit single by Robbie Williams, Trippin. He downloads the song to his phone. Vikraams suddenly remembers that he has an appointment with the doctor and Christina quickly downloads Vikraams music library list. Vikraam leaves the group and Christina browses to the list she just received to see if Vikraam has something new she can download later. Joey is also interested and downloads Vikraam's list from Christina which results in an update of the last list he received from Vikraam. Joey also downloads a song from Christina before they decide to exit PeerShare and go to lecture.

17.2 Goals

By analyzing the scenario above, we can extract a handful of goals that the application must fulfill. The goals are:

1. The application must allow several users to connect to each other
2. The application must support transfer of files and file lists between peers
3. The application must be able to store the files in an organized manner
4. The files must be accessible from other applications on the phone

17.3 Functional requirements

Table 17.1 lists the functional requirements we have set out for this application. The functional requirements meets the goals mentioned above.

Requirement	Text
FR1	All participants must have a file that contains a list of shared files stored on their phone
FR2	All participants must be able to download a list of files from each other
FR3	A user must be able to navigate through a list of files and download the desired file
FR4	All downloaded files should be stored in a default download directory
FR5	The application must have a settings menu
FR5.1	The user must be able to set the default download directory
FR5.2	The user must be able to specify which directories he or she wants to share
FR5.3	The user must be able to set a nickname
FR6	The application must support all kind of files
FR7	The user should be informed of what is going when operations are expected to take a few seconds. A progressbar should also be shown during this process

Table 17.1: *Functional requirements for PeerShare*

17.4 Usecase description

The use-case diagram, see Figure 17.1, illustrates how the functional requirements relates to each other and the user of the application.

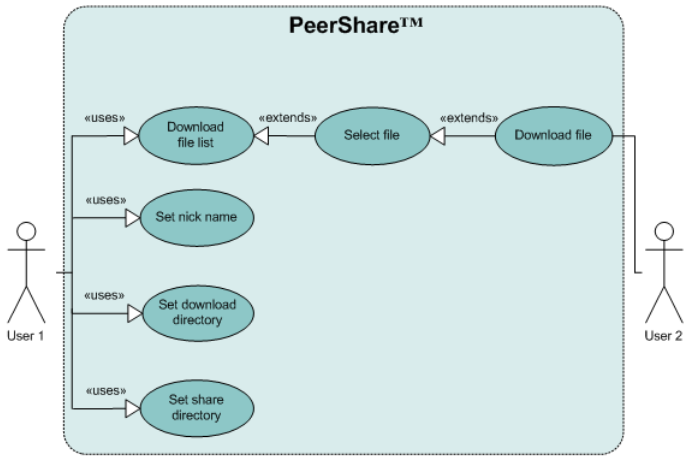


Figure 17.1: *Use case model for PeerShare*

17.5 Quality requirements

Quality requirements are just as important as functional requirements and it is easy to get carried away and just create an application with a lot of functionality without regard to the non-functional aspects. This section contains a few quality scenarios we find important.

17.5.1 Usability

U1 – Few user prompts	
Source of stimulus	User
Stimulus	User access the local filesystem through the PeerShare GUI
Environment	Runtime
Artefact	PeerShare application
Response	The user should not be prompted for access every time the application tries to access the local filesystem
Response Measure	The application should never ask the user for this sort of permission

Usability scenario U1 is important to ensure usability. It is quite annoying to be prompted for the same action over and over.

17.5.2 Testability

T1 – Log file	
Source of stimulus	Developer
Stimulus	Developer wants to know if exceptions are thrown
Environment	Runtime
Artefact	PeerShare application
Response	Application should catch all exceptions and write the appropriate error message on the mobile phone display
Response Measure	These messages should be easily switched on and off

Testability scenario T1 is very helpful for the developer. Mobile phones have small displays and error checking is very hard to do during runtime if special considerations have not been made. Therefore, by allowing the developer to see error messages during runtime, finding bugs will take less time.

17.5.3 Modifiability

M1 – Download a whole folder	
Source of stimulus	Developer
Stimulus	The developer wants to add the functionality of downloading a whole folder
Environment	Design time
Artefact	PeerShare application
Response	The change should be easy to accomplish with no ripple effects
Response Measure	Downloading multiple files would require multiple messages to be sent. The change should be accomplished within 5 hours.

The functional requirements in Section 17.3 only specifies that single files can be downloaded from other peers. A possible expansion of the application would be to allow the user to download whole folders, instead of single files. Modifiability scenario M1 describes this expansion.

17.5.4 Availability

As mentioned earlier, it is very important to pay special attention to availability in the PeerShare application. The J2ME Bluetooth environment does not allow a node to function as a slave and a master at the same time. Therefore, if one slave node wants to download a file from another slave node, the master node will have to function as a intermediary. See Figure 17.2 for illustration of the file transfer process. All

three nodes can at any time suddenly break connection. This needs to be handled differently depending on which node has disconnected. If one node looses connection, the two others are also affected.

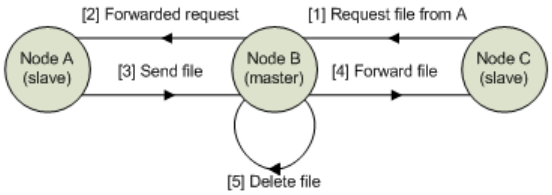


Figure 17.2: *File transfer in PeerShare*

A1 - Broken connection during file transfer	
Source of stimulus	File transfer
Stimulus	One of the nodes in 17.2 disconnects
Environment	Runtime during file transfer
Artefact	All nodes in 17.2
Response	The user most be informed of the occurred event. The application must not hang.
Response Measure	A couple of seconds should be given to see if the connection can be restored. If this is not possible, the operation should be aborted on the affected nodes and they should continue to operate as usual. If the master node disconnects, all the other slavenodes must automatically be disconnected

We have chosen not to include a performance scenario here, because it is highly dependant on the mobile phones which the application is deployed on. Also, since the Bluetooth API is restricted to the hybrid P2P model (see Section 6.1), the application is dependant on an effective master node. It is therefore optimal that the master node is the mobile phone with the most resources available.

CHAPTER 18

Dependencies

In order for the PeerShare application to work, the environment it runs in must satisfy certain criteria. Even though Java is meant to be platform independent, there are a lot of different J2ME implementations on todays mobile phones. J2ME comes with several optional packages and PeerShare uses two of these packages. Not all phones that support the J2ME library supports all optional packages.

18.1 Packages

PeerShare has one more dependency than PeerQuiz. PeerShare will not run without JSR 75 support. This application uses the following packages:

- *J2ME library* - The standard J2ME library.
- *Peer2Me.jar* - The Peer2Me framework.
- *jsr082.jar* - The Java APIs for Bluetooth, also called JAWBT.
- *jsr75.jar* - PDA Optional package for accessing PIM data (Personal Information Management) and file system
- *kxml2-min.jar* - An open source xml pull parser designed for use on devices with limited resources.

The Peer2Me framework uses the J2ME library and the jsr082.jar package. This application uses two additional packages. The kxml2-min.jar package can be run on most mobile phones, jsr75.jar can not.

18.2 Supported mobile phones

Since we are using the package jsr75 in order to access the filesystem on the mobile phone, there are currently only a handful of mobile phones which supports this package. PDAs were the first devices that supported this package. The newest Sony Ericsson mobile phones supports the use of jsr75, see table 18.1. The phones that supports jsr75, also supports JSR 82 (JAWBT) package which is also needed by the Peer2Me package.

Manufacturer	**Model**	**JSR 75 support**	**JSR 82 support**	**At our disposal**
Sony Ericsson	W900i	Yes	Yes	No
Sony Ericsson	W800i	Yes	Yes	Yes
Sony Ericsson	W600	Yes	Yes	No
Sony Ericsson	W500	Yes	Yes	No
Sony Ericsson	K750i	Yes	Yes	Yes
Sony Ericsson	K600	Yes	Yes	No
Sony Ericsson	V600	Yes	Yes	No
Sony Ericsson	z520	Yes	Yes	No
Sony Ericsson	p900	Only PIM API	Yes, but with bugs	Yes
Siemens	s65	Only PIM API	Yes	Yes

Table 18.1: *Supported mobile phones*

CHAPTER 19

Design

This application uses the same architectural principles as the PeerQuiz application. PeerShare comprises of three packages: *peershare*, *peershare.gui* and *peershare.util*. Each package is designed for a specific purpose which this chapter will elaborate further. The main class, *PeerShare* is located in the package *peershare*.

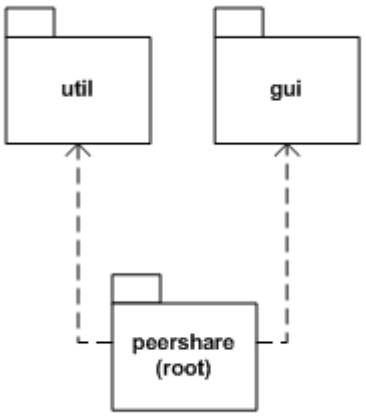


Figure 19.1: *Package diagram for PeerShare*

19.1 The peershare package

The *peershare* package contains the Midlet class, *PeerShare*, which is the primary class of the application. It also contains an interface and some other classes that are used both by *PeerShare* and some of the classes in the *peershare.gui* package.
This package is illustrated in Figure 19.2.

Constants: An interface that contains constants that are used by several classes.

PeerShare: This is the primary application which extends Midlet. Both slave and master nodes run this class. A variable is set in this class which determines whether the application should run as master or slave.

ShareNode: This class adds additional information to the *no.ntnu.idi.ntnu.mowahs.project.domain.Node* class. This information is used to store information about the node's shared folders and files. It also contains an *Image* that is associated with the node.

ShareNodes: Extends *java.util.Vector* and keeps a collection of *ShareNode*. This class is used so that each node in the network can keep track of each other.

ShareObject: A simple class that determines if a shared object is a folder or a file.

19.2 The peershare.gui package

This package contains the classes which extends a *Displayable* object such as *Form* and *Image*, which can be displayed to the user, hereafter referred to as a *screen*. All user input is handled by the classes in this package. *BarItem* is the exact same class that is used in PeerQuiz, see Section 13.3. *CommunicationScreen* is also almost exactly like the *CommunicationScreen* class used in PeerQuiz, see 13.3, but with some changes.
This package is illustrated in Figure 19.3.

BarItem: A class that can be used as a progressbar. This class is used so that the user can know that something is happening in the background and that the application has not stalled.

CommunicationScreen: A screen that the user is presented with when a communication link is being established between the slaves and the master. Information about what is going on is displayed here along with the *BarItem*.

DirectoryBrowserScreen: A screen that lets the user select which folders to share and where downloaded files should be stored.

MainMenuScreen: The first screen that the user is presented with. Gives the user three choices: host, join and settings.

NickNameScreen: A screen that lets the user select a nickname.

PeerBrowserScreen: A screen that the lets the user browse through all the nodes which has joined the same PeerShare group and download shared files.

SettingsScreen: Current implementation only presents the user with one adjustable setting: to change download and upload folder. The idea is that more settings can be added later.

19.3 The peershare.util package

This package contain classes that can do operations on the mobile phones recordstore and filesystem. It also contains a class that can parse XML and logging functionality. *PeerShareRecordStore* has the same contents and structure as *QuizRecordStore*, described in Section 13.5, but with less methods and some small modifications. *ShareLog* and *ShareLogMessage* are the same as *QuizLog* and *QuizLogMessage* used in PeerQuiz, see Section 13.5.
This package is illustrated in Figure 19.4.

FileHandler: A class that can read and write files to the local filesystem. This class also contains recursive methods that creates an XML file which contains an hierarchical structure of folders and files.

PeerShareRecordStore: This class reads and stores application settings such as nickname, folder settings and information about shared files.

ShareLog: Extends *java.util.Vector* and stores a collection of *ShareLog*. This class is used to log application events and can be viewed on the mobile phone in runtime for testing purposes.

ShareLogMessage: Is used by ShareLog to store single application events.

XMLTool: A class that reads XML files, in this case, the XML file created by the *FileHandler*. The class can be used to read both local and externally created XML files. It can also alter the content of an XML string and return the altered string. The class uses an XML pull Parser.

19.4 Class Diagrams

This Section contains the class diagrams for all the packages in PeerShare.

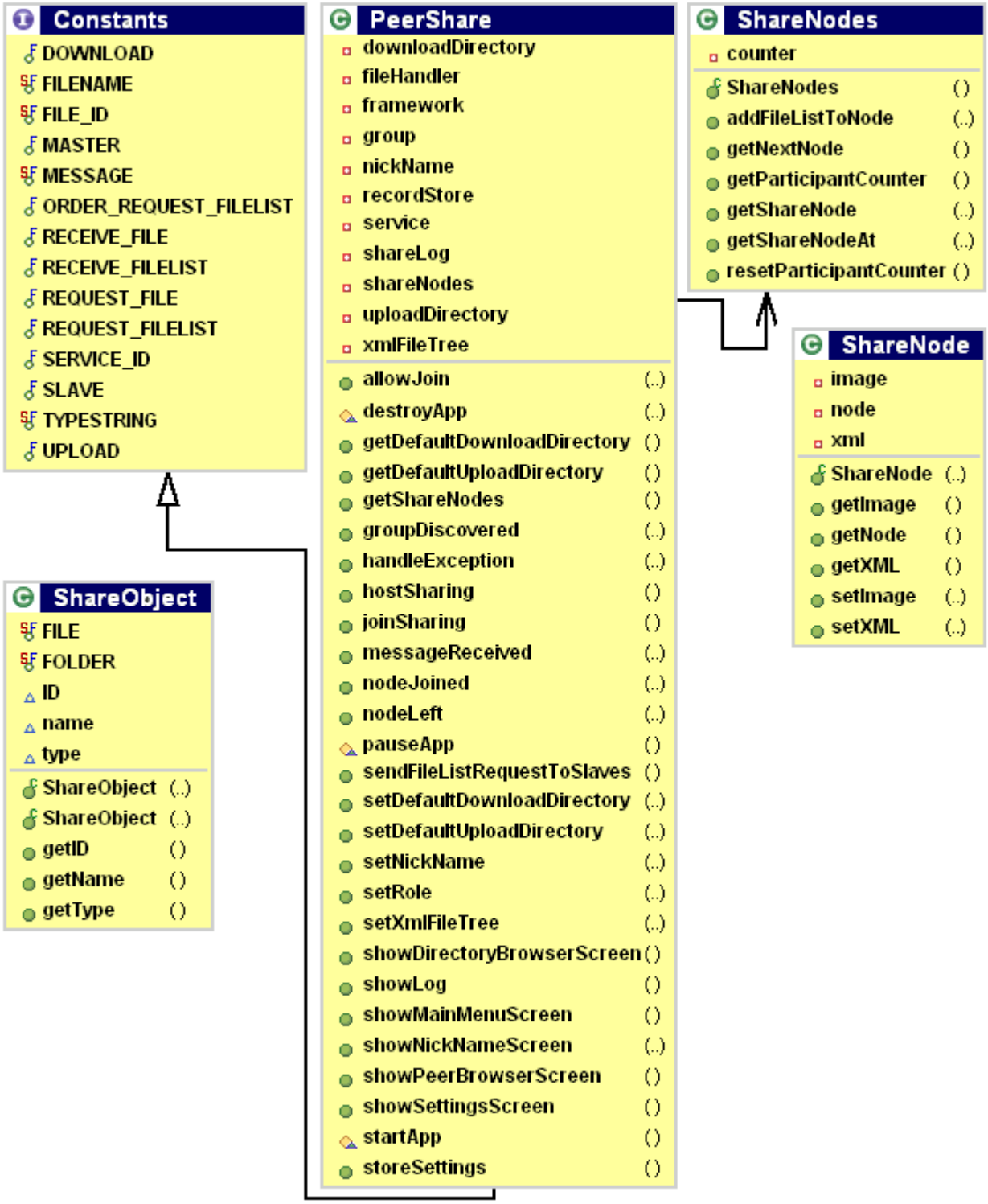


Figure 19.2: *The peershare package*

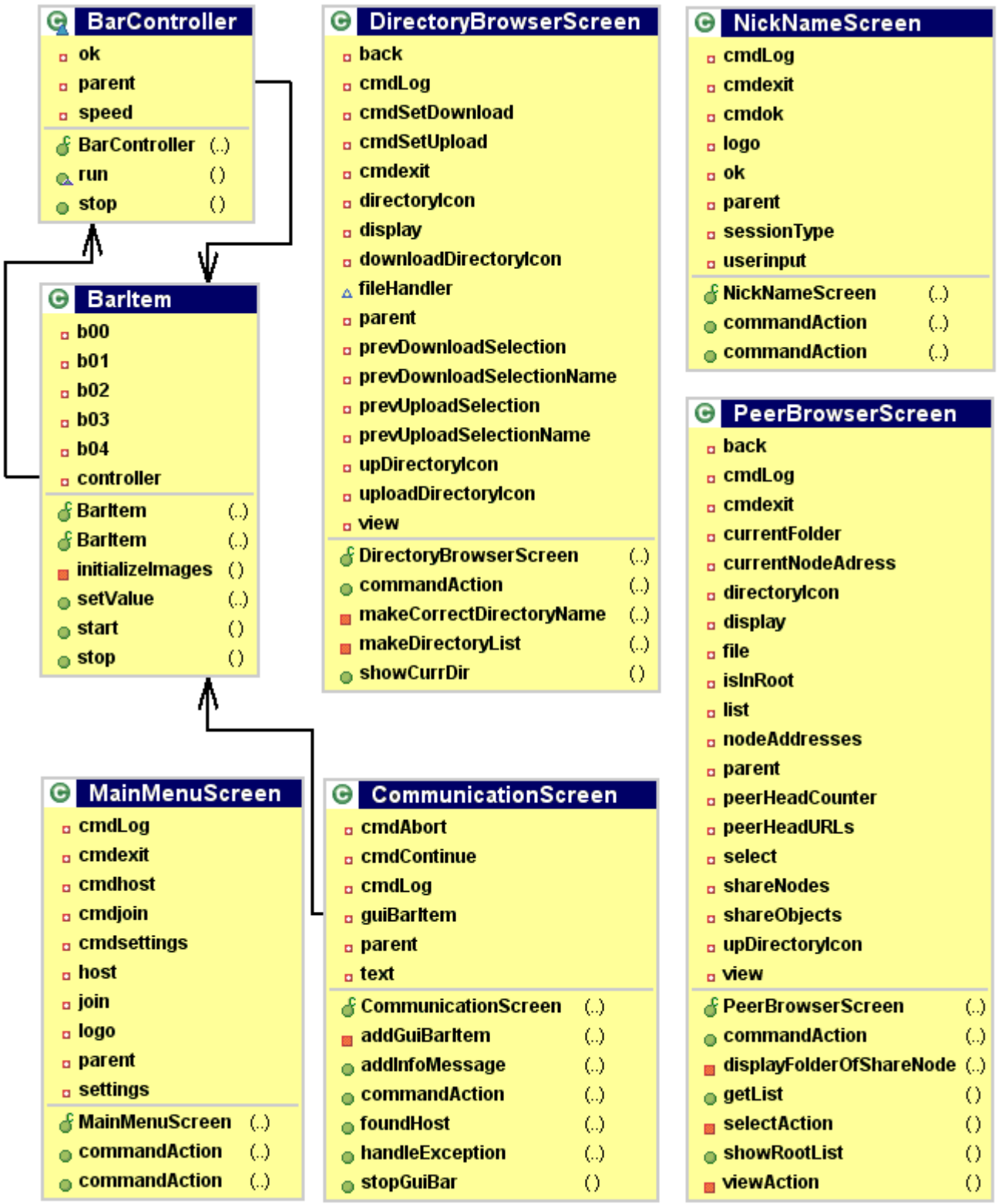


Figure 19.3: *The gui package*

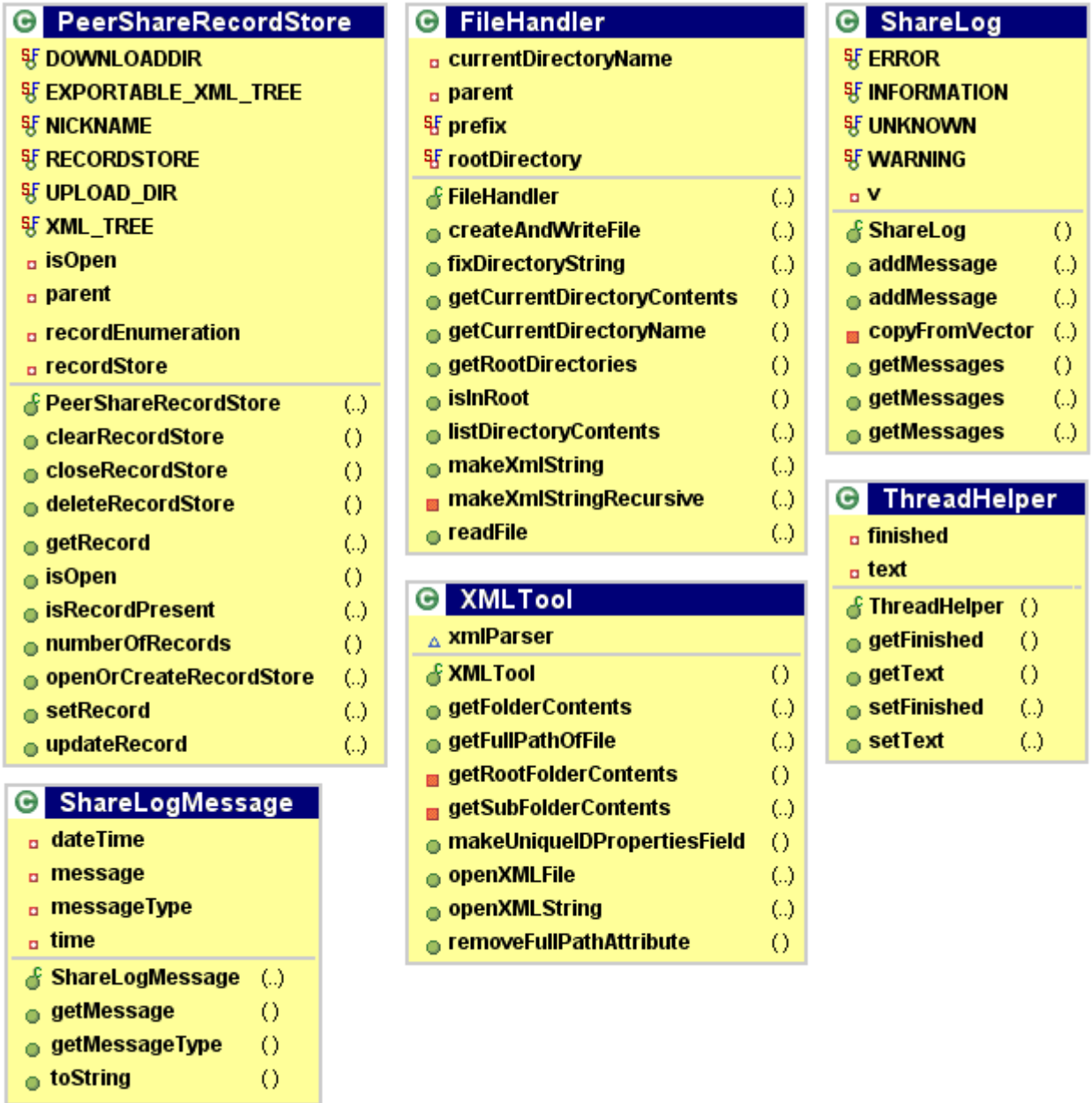


Figure 19.4: *The util package*

19.5 Program flow

This Section describes the flow in PeerShare. When starting the application for the first time, the user needs to select download and upload folders. Then the user needs to decide if he wants to function as a master(host) or a slave(join) node. There can only be one master, and all slaves need to join before the master hosts the game. This may seem awkward, but this is the way the Bluetooth API works. After all nodes have joined and a master has been selected, the nodes starts exchanging files automatically and then the filesharing can begin. Figure 19.5 shows an activity diagram of the program flow which can be divided into two main phases: "Setting up the network" and "transferring files".

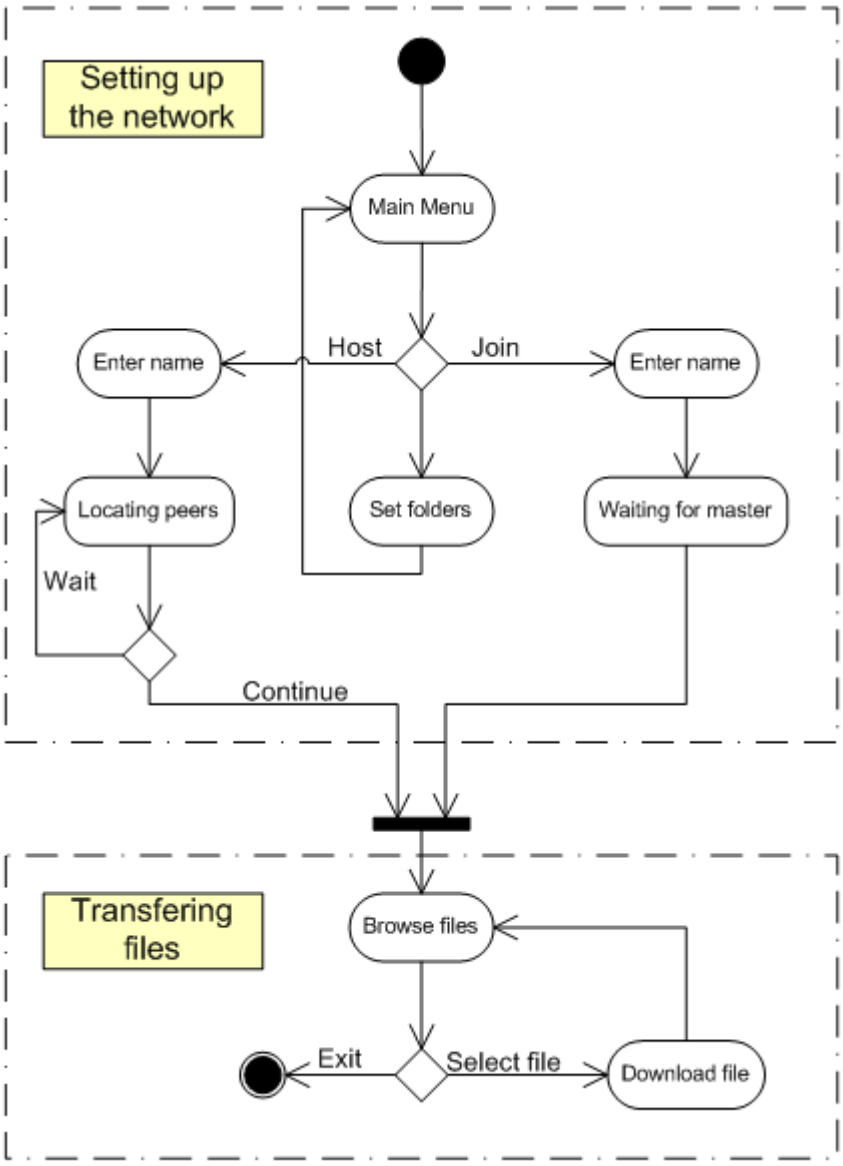


Figure 19.5: *Activity diagram of PeerShare*

CHAPTER 20

Implementation

This Chapter describes the implementation details of the PeerShare application. PeerShare executes operations on the mobile phone's filesystem, which is described in detail. These operations creates or uses xml files in order to sort the information retrieved from the filesystem. The kXml parser used in PeerQuiz is also used PeerShare. The graphical user interface concepts that were used in PeerQuiz, has also been used in PeerShare.

Communication an essential part of this application. PeerShare does not separate master and slave in two different classes as the PeerQuiz application does. Therefore, the Peer2Me initialization is done in one class and not two. Also, since PeerShare is a filesharing application, details around how messages are sent is described. Lastly this Chapter describes how application data is stored and some code statistics.

20.1 FileConnection APIs

Only a few mobile phones support filesystem access through J2ME as mention in Chapter 18. We have created one class for this purpose, *peershare.util.FileHandler*. This class imports necessary classes from jsr75 shown in Listings 20.1.

Listing 20.1: *Imports in the FileHandler class, from* `FileHandler.java`

```
3 import javax.microedition.io.Connector;
4 import javax.microedition.io.file.*;
```

In order to use this class, one have to create an instance of it and send a reference to a PeerShare object as a parameter to the FileHandler's constructor. When the application needs to retrieve all folders in a folder, a call to the fileHandler's `listDirectoryContents(..)` method is done as illustrated in Listings 20.2. We figured that it was unnecessary to return the files in a folder when specifying the download and upload folders in the application. A possible expansion of the application would be to let the user decide if files should also be shown. This does not require a lot of changes.

Listing 20.2: *Getting content of directory, from* `DirectoryBrowserScreen.java`

```
154         Enumeration e;
155            e = fileHandler.listDirectoryContents(directory,true);
```

If the specified folder is a valid folder and not the root directory the code listed in Listings 20.3 will be executed. An enumeration list with the content will then be returned.

Listing 20.3: *Connecting to filesystem and retrieving contents, from* `FileHandler.java`

```
150   FileConnection conn = null;
151   try
152   {
153       conn = (FileConnection) Connector.open( currentDirectoryName );
154       if(conn.isDirectory())
155       {
156        if(!changeCurrentDircetory)
157      currentDirectoryName = prevDirName;
158        return conn.list();
159       }
```

When the user selects a folder he or she wants to share, all files and subfolders in that folder is organized in an xml string. In order to traverse through a folders subfolders, subsubfolders and so on, we created a recursive method which is a private method in *FileHandler* called `makeXmlStringRecursive(..)`. This method is indirectly called through the `makeXmlString(..)` method as illustrated in Listings 20.4.

Listing 20.4: *Calling the recursive method, from* `FileHandler.java`

```
233  public String makeXmlString(final String directory)
234  {
235   final ThreadHelper textHolder = new ThreadHelper();
236   //Create an inner Thread to avoid deadlocks
237   new Thread(new Runnable()
238   {
239    public void run()
240    {
241     textHolder.setText(makeXmlStringRecursive(directory));
242     textHolder.setFinished(true);
243    }
244   }).start();
245
246   //Just waiting for the thread to finish before moving on..
247   while(!textHolder.getFinished())
248   {}
249   return textHolder.getText();
250  }
```

Every time the application accesses the filesystem, the operation must be done in a separate thread to avoid potential deadlock. We have chosen to create inner threads in the *FileHandler* class in order to "hide" this from the rest of the application. Classes that uses the *FileHandler* class does not need to know that the operations are performed in separate threads.

20.2 Xml parsing

The *FileHandler* class returns an xml-document represented as a `String`. The xml document follows the DTD (Data Type Definition) illustrated in Listings 20.5.

Listing 20.5: *DTD for example.xml, from* `rules.dtd`

```
1  <!ELEMENT dir ( dir | file )*>
2  <!ATTLIST dir path CDATA #REQUIRED>
3  <!ELEMENT file EMPTY>
4  <!ATTLIST file
5   name CDATA #REQUIRED
6   id CDATA #REQUIRED
7   fullPath CDATA #IMPLIED>
```

The application stores two types of xml files; one for local reference and one that is distributed to the other nodes. The difference is that the local xml file includes the attribute `fullpath` in the `file` element. The `id` attribute contains a unique id so that each file is associated with a unique number. When a node receives a request for a file, it receives the id and can easily fetch the correct file from the filesystem using the locally stored xml file. An excerpt from a sample xml file is illustrated in Listings 20.6.

Listing 20.6: *Sample xml file, from* `example.xml`

```
<dir path="shared/">
 <dir path="shared/articles/">
  <file id="1" fullpath="file:///root1/shared/articles/Programming in java.pdf"
   name="Programming in java.pdf"/>
  <file id="2" fullpath="file:///root1/shared/articles/Secrets of C#.txt"
   name="Secrets of C#.txt"/>
 </dir>
 <dir path="shared/big text files/">
```

The class that handles the xmlparsing is *peershare.util.XMLTool*. This class uses the kXmlparser mentioned in Section 9.4.4 to read the xml file. XMLTool is constructed so that it can take either a String containing the xml or the filename of an xmlfile that is compiled with the application. Reading the xml is done in the same way as for PeerQuiz, see Section 14.1.

20.3 GUI - Graphical user interface

The graphical user interface implementation follows the same pattern as in *PeerQuiz*. The difference is that in *PeerShare* we have used the class *javax.microedition.lcdui.List* as a graphical item in addition. This class is used when the user is browsing through his or others files.

20.4 Communication

This part of the application utilizes many of the functionalities offered by the Peer2Me framework. All communication is done through the *PeerShare* class regardless of the node's role, slave or master. In *PeerQuiz* we had two classes, *QuizMaster* and *QuizSlave*, because the master had more responsibility than in *PeerShare*.

20.4.1 Initialization

Firstly all nodes that want to join the network must do so before the master start hosting. This may sound awkward, but as mentioned in the prestudy, it is the master who searches for the slaves and not vice versa. In order for a slave to join the network, PeerShare runs the code in Listings 20.7. The PeerQuiz application runs the same code, but in two different classes; *QuizMaster* and *QuizSlave*.

Listing 20.7: *Slave - Initializing components, from* `PeerShare.java`

```
193   service = new Service(SERVICE_ID);
194   framework = Framework.getInstance("" + nickName, "PeerShare application" ,
195     "no.ntnu.idi.mowahs.project.bluetooth.network.BluetoothNetwork");
196   framework.init();
197   framework.setGroupDiscoveryListener(this);
198   framework.setMessageSubscriber(this);
199   framework.setExceptionHandler(this);
200
201   framework.registerService(service);
```

After all slaves have done this, the master needs to initialize the framework as well and start looking for slave nodes. Listings 20.8 and 20.9 shows how this is done.

Listing 20.8: *Master - Initializing components and start search, from* `PeerShare.java`

```
156   service = new Service(SERVICE_ID);
157   framework = Framework.getInstance("" + nickName, "PeerShare application" ,
158     "no.ntnu.idi.mowahs.project.bluetooth.network.BluetoothNetwork");
159   framework.init();
160   framework.setGroupDiscoveryListener(this);
161   framework.setMessageSubscriber(this);
```

```
162   framework.setExceptionHandler(this);
163   framework.setNodename(nickName);
164
165   group = new Group();
166   group.setMaster(framework.getLocalNode());
167   group.setMonitor(this);
168   service.setGroup(group);
169   group.setService(service);
170
171   framework.registerService(service);
```

Listing 20.9: *Master - Starting search, from* `PeerShare.java`

```
213   framework.startGroupSearch(service);
```

When a master discovers a slave, the method `groupDiscovered(..)`, which is a required method by the Peer2Me framework, is invoked at the slave node. The slave starts monitoring the group created by the master and joins the network, see Listings 20.10.

Listing 20.10: *Slave - joins group, from* `PeerShare.java`

```
325   group.setMonitor(this);
326   framework.joinGroup(group);
327   shareNodes.addElement(new ShareNode(group.getMaster(),""));
```

Every time a new slave node is found, the `nodeJoined(Group group, Node node)` is invoked by the master and the slaves and a reference to the node that joined the network is stored. The user that runs the master node decides when enough slaves has joined the network and presses "Continue" on his mobile phone.

20.4.2 Setting up the filesharing network

After the master has found all the slaves, the nodes need to exchange filelists. Each node has an xmlfile the can be sent to other nodes as mentioned in Section 20.2. We decided to put the Peer2Me framework to a stress test to see if the messagesystem was able to handle many messages at the same time. If the group consists of one master and two slaves, 14 messages will be exchanged in total between the nodes.

The PeerShare application has been tested with 1 master and 4 slaves, which resulted in 44 messages being exchanged, just to receive the filelists. Figure 20.1 shows how one master and two slaves will exchange messages.

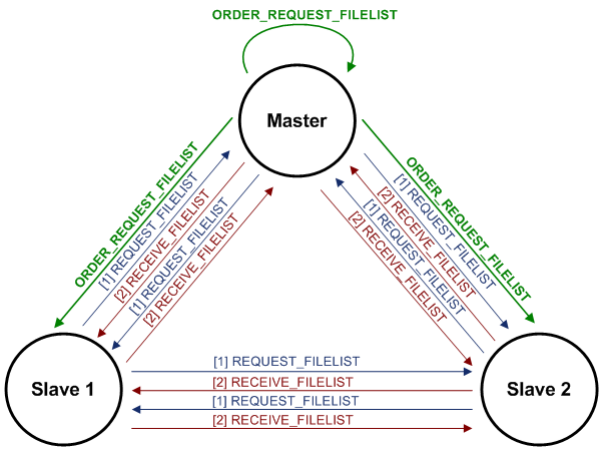


Figure 20.1: *Messages exchanged to distribute filelists*

In order to lower the amount of total messages sent, the method illustrated in Figure 20.2 can be used to share the filelists. Here, the master collects all the lists and then distributes the list to the slaves afterwards.

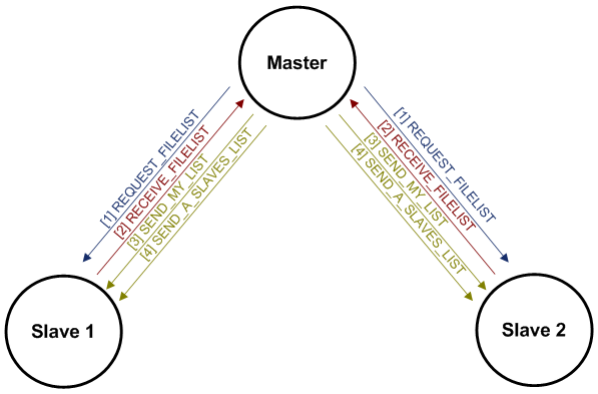


Figure 20.2: *Messages exchanged to distribute filelists*

20.4.3 Downloading files from others

After all nodes have received everyone's filelist, the sharing can begin. From this point on, the master has no special tasks and the GUI is exactly the same on the master and the slaves. The user is presented with a list of the other users represented as small heads. When the user clicks a head, an associated filelist will be shown in the same way as when selecting upload and download folders (see Section 20.3). When a user selects a file to download, a message is sent to the node that has the file. The message contains the fileID and the name of the file. When the node receives a file request, it uses the XMLTool to find out which file that is requested, reads the file and sends the contents back to the sender. This is illustrated in Figure 20.3.

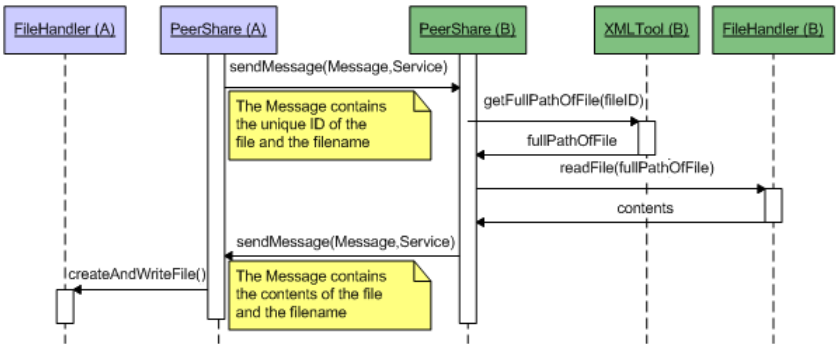


Figure 20.3: *Requesting and sending file*

20.4.4 The truth about Peer2Me communication

When using the Peer2Me framework to send messages, the developer has to specify which node to send the message to before sending it. Even though it seems that the message is sent directly to the recipient, it always passes through the master node. The Peer2Me framework hides this from the developer which means that the developer must carefully consider how to distribute a large amount of messages if he

or she wants the application to become as efficient as possible. We have not aimed to create the most efficient algorithm for exchanging filelists in PeerShare, but rather tested the Peer2Me functionality.

20.5 Storing application data

The Peer2Me framework offers a functionality to store objects in the mobile phones recordstore. For this purpose the object to be stored must implement the *no.ntnu.idi.mowahs.project.utilPersistent* interface. In order to store the object in the recordstore, the *no.ntnu.idi.mowahs.project.utilPersistentManager* class must be used. After several attempts, we have been unable to make this work. Therefore, we created our own way of storing information in the recordstore. PeerShare uses the *peershare.util.PeerShareRecordStore* class to store information. When the application exits, it stores nickname, download directory, upload directory and two versions of the xmltree. This is done to avoid the tedious operation of selecting folders when the application starts.

20.6 Code statistics

In order to illustrate the size of the application, we have included some code statistics for the PeerShare application. See Table 20.1.

Aspect	**LOC**	**# of classes**	**Avg. # of methods**	**Avg lines per method**
peershare.util	767	6	23.8	12.17
peershare.gui	716	8	13.25	12.27
peershare	544	4	19	5.16
Total	2027	18	18.056	9.36

Table 20.1: *PeerShare code statistics*

CHAPTER 21

Testing

This chapter presents the covered functional and non-functional requirements set out in Chapter 17.

21.1 Test of functional requirements

PeerShare has been tested in the same environments as the PeerQuiz application:

Sun's Wireless Toolkit 2.2 with included emulator running on an x86 computer using Windows XP SP2 and Java version 1.5.0_04.

Mobile phones on test subjects. Sony Ericsson W800i and Sony Ericsson K750i.

Table 21.1 lists the functional requirements set out in Chapter 17, a couple of new requirements and if the application supported the requirement.

21.1.1 Comments on the implementation of the requirements

Requirements FR1, FR2, FR4 and FR5 could not be tested thoroughly on the emulator. It is possible to run multiple instances of the emulator, but only one instance has access to the emulated filesystem and recordstore. However, we did manage to transfer dummyfiles and dummyfilelists [1], but this did not utilize all application modules. This meant that we had to compile, deploy and run the application on at least two real phones in order to test these functionalities. The process was time-consuming because it had to be repeated a couple of times before we got the application working properly.

FR2 The application automatically downloads all the filelists from all the nodes. The application does not currently support refreshing filelists. Implementing this functionality would not take more than 30 minutes if the developer is familiar with the application.

FR8 The Peer2Me framework only supports transfer of messages that are in ASCII format. It is possible to transfer binary files between Bluetooth nodes with J2ME, but not with the use of the Peer2Me framework.

[1] Dummyfile/dummyfilelist - instead of a real file/filelist, we transferred a long string value that was hardcoded in the application

Requirement	Text	New requirement	Result
FR1	All participants must have a file that contains a list of shared files stored on their phone	No	Pass
FR2	All participants must be able to download a list of files from each other	No	Pass
FR3	A user must be able to navigate through a list of files and download the desired file	No	Pass
FR4	A user must be able to download a file from another user	No	Pass
FR5	All downloaded files should be stored in a default download directory	No	Pass
FR6	The application must have a settings menu	No	Pass
FR6.1	The user must be able to set the default download directory	No	Pass
FR6.2	The user must be able to specify which directories he or she wants to share	No	Pass
FR6.3	The user must be able to set a nickname	No	Pass
FR7	The user should be informed of what is going when operations are expected to take a few seconds. A progressbar should also be shown during this process	No	Pass
FR8	The application must support transfer of all kind of files between nodes	No	Not Possible
FR9	The application should store application data such as nickname, download folder, upload folder and information about shared files when the user exits the program	Yes	Pass
FR10	The application should automatically load saved application data when it starts	Yes	Pass

Table 21.1: *Fulfillment of functional requirements for PeerShare*

FR9 We found out during testing that it was very time-consuming to specify nickname, download folder and upload folder every time we started the application. The information is currently being stored in the recordstore.

FR10 If FR8 should be useful, the application data must also be loaded.

21.2 Test of quality requirements

U1 - Few user prompts - Not implemented. After testing the applications developed by Sars and Norum in [22], we wanted to avoid that the application prompted the user every time it used the Bluetooth API. The same thing happened when were experimenting with the FileConnection API. This usability requirement proved to be unresolvable in the programming code. In order to avoid user prompt popups when the JAWBT is invoked, the user has to manually configure his mobile phone to automatically accepts that the JAWBT can operate freely without the users approval at runtime. The case is not the same for the FileConnection API which requires that the Midlet is digitally signed. The signature must also be verified by a third party such as VeriSign. A standard code signing certificate from VeriSign costs $400, which we do not possess. See VeriSign's webpage for more information [38].

T1 - Log file - Implemented. The programmer has to manually specify which messages should be stored in the log. The log can be easily toggled in runtime on the mobile phone, but is not stored when the

application exits. In order to store the file, the developer can use the `peershare.util.FileHandler` class and write the file to any location he wants. We did not choose to do so as viewing the log during runtime proved to satisfy our needs.

M1 - Download a whole folder - The application's architecture makes it possible to download a whole folder. All folders and associated files are stored in an xml file. In order to implement this functionality, the user who requests the folder, will only need to send one message. The owner of the files will have to send one Message for each file.

A1 - Broken connection during file transfer - Not implemented due to errors with the Peer2Me framework. Peer2Me "hides" that a message is sent using the master as an intermediary. Furthermore, the Peer2Me framework has a method called `nodeLeft(Node node` which is supposed to tell the nodes in the group when a node is leaving. We have tried to use this method, but it does not seem to work. Therefore, in order to fully implement this quality requirement using the Peer2Me framework, we need to find out why this method is not working. Currently, when a node asks for a file, it just waits for it to arrive and the user can do other operations in the meantime. It does not matter if a node asks for several files from different nodes and they arrive in different order than requested.

Part VI

Evaluation

CHAPTER 22

Answers to research questions

This Chapter will answer the research questions presented in Chapter 3.1.

22.1 Developer questions

Question 1

Can a developer with some experience in Java, easily adopt the Peer2Me framework as a utility for developing applications for mobile phones?

We have made some small modifications to the questionnaire that Sars and Norum used in [22]. We have answered the questionnaire and the answers can be found in Appendix C. By comparing the participants in [22] with ourselves, we see that we have approximately the same programming experience. The only difference is that Tommy has previously developed Bluetooth applications, but not using J2ME.

When it comes to the understanding of the domain concepts of Peer2Me we see a clear difference. The participants seem to have a better understanding of some of the concepts than we do. The questionnaire is purely subjective and the small applications developed by the participants did not involve close inspection of the underlying logics in the framework. Since we have inspected the framework more closely, we may have started to question the naming of the different logics in the framework.

Question 1 cannot be answered in a simple yes/no fashion. We do not agree that it is easy to start using the framework. In order to use the framework, it must be initiated. One must also create a group, service, set listeners, messagesubscribers, exceptionhandlers, monitors etc. This must be done differently depending on the role of the node that runs the application, the role can either be slave or master. If we want to have an application that can run as master during one session and a slave during another session, we have to take this into account when developing the application. There are also plenty of methods that are required by all the interfaces the MIDlet has to implement from the Peer2Me framework. It is time-consuming to find out how to write these methods. We also spent a great amount of time getting used to the J2ME environment and compiling and deploying applications on mobile phones. This is not Peer2Me's fault, but rather our lack of experience with J2ME, which we now have gained.

However, there is no doubt that the framework saves the developer a lot of codelines, but before using this, it has to be 100 % reliable. This is not the case today.

Question 2

Which bugs exists in the framework and what kind of impact do they have for development?

During the development of PeerQuiz and PeerShare, we have come across several bugs that is of different severity, but they all effect the usage of the framework. The answer to this research question contains the bugs found and how these affect the framework. The answers to question 3 includes projected time to fix these bugs and what outcome this will lead to.

Description of bug	The method `nodeLeft(..)` does not function or respond at all
Found in class	*...framework.GroupMonitor*
Affected classes	*...domain.group*
Priority	High
Est. time to correct	30 minutes

Comments:
This bug concerns a specific method that is called when a node leaves the network **on purpose**, and not by accident. This is very ambiguous, since it at first eyesight appears to be a method that is called whenever a node leaves the group, regardless by the condition. We will discuss how this can be improved in the third research questions.

Description of bug	The messages cannot contain a carriage return (\n) and the message is ended when this occur.
Found in class	*...domain.Message*
Affected classes	No other classes
Priority	High
Est. time to correct	4 hours

Comments:
This bug originates in an algorithm in the method `parseBytes(..)` in *domain.Message*. The messagetext is converted to a bytestream until it detects a carriage return (\n), which means you **cannot** send multiple lines of text in one messagepart. This is a silly bug, and the detection of a message-ends should be done in a more sophisticated manner.

Description of bug	A slavenodes name is not possible to retrieve when it is entered directly in the constructor when creating a slavenode
Found in class	*...framework.Framework*
Affected classes	No other classes
Priority	Low
Est. time to correct	30 minutes.

Comments:
This is a weird bug, since it only appears on the slavenode. When *Framework* is initialized on a slavenode, a nodename is supplied to the constructor, but it is not display when the slavenode is discovered by the

masternode. A null-pointer occurs. If you explicitly call the method `setNodename(..)`, the nodename is stored and displayed to the other nodes.

Description of bug	Often `handleException(..)` does not catch underlying exceptions thrown by the other parts of the framework, nor by the application
Found in class	*...framework.Framework*
Affected interface	*...framework.HandleException*
Priority	Medium
Est. time to correct	1 hour.

Comments:
The *handleException* method required by the classes that uses the Framework, seems to malfunction in certain situations. As an example, it is not possible to throw exceptions to be caught by the class implementing *ExceptionHandler*, and often *handleException* is not invoked in cases of frameworkexceptions either.

Description of bug	It is not possible to compile the framework without the test-applications
Found in class	N/A
Affected interface	N/A
Priority	High
Est. time to correct	This is a quite time-consuming task. Unable to estimate the time it takes to fix this bug.

Comments:
During the testing phase of the framework, several debug methods and test applications were created. These are implemented into the framework, and is dependent of each other. There is a strong dependency between the different elements in the framework and its test applications. This gives great overhead and poor exploitation of available space.

Question 3

How can we improve the framework?

An important part of the framework evaluation is to determine points of improvement and new functionality that can be added. This answer discusses bugs found in research question 2 worth fixing and how they can be fixed. Other mprovements are also discussed in research question 4.

Element	Messaging system
Classes affected	*...domain.Message* and *...framework.Framework*
Est. time to change	This is a quite complicated tasks estimated to take several weeks
Projected positive outcome	Adds the ability to send binary data, not only text
Projected negative outcome	Network traffic increases due to the possibility of sending great amounts of data.

Comments:
This improvement is considered as one of the most important features to be implemented. Implementing this feature will give the developer incentive to develop richer and useful applications. The possibility

to send and receive binary data such as pictures and other media files, will make the framework more flexible and widen the usage area.
The applications we developed will benefit from this improvement. For example in PeerShare, the users could exchange MP3 audio files, and in PeerQuiz a question could contain a picture or sound. One a more technical level, objects can be sent as messages and the receiver can cast the object to the correct type.

Element	Connection loss detection
Classes affected	N/A
Est. time to change	This is a quite complicated tasks estimated to take one week
Projected positive outcome	Adds a connection loss detection system
Projected negative outcome	Network traffic increases due to more transmissions

Comments:
This improvement will add valuable availability to the framework. Today, one of the biggest problems related to the connections of Bluetooth nodes is the ability to detect if one or more nodes are ascent. If a node has gotten out of reach from the masternode, the masternode does not get any message that a node is lost. This can be solved by implementing a timeout or ping/echo functionality in every application using the Peer2Me framework, but should instead be a part of the framework, not the applications. The only negative outcome would be the increase in network traffic.

Element	Utilizing the Java Recordstore
Classes affected	*...util.PersistenceManager* and *...util.Persistence*
Est. time to change	Three days
Projected positive outcome	Creates a better way of storing data in the recordstore
Projected negative outcome	None

Comments:
Today the Peer2Me framework is utilizing the Java Recordstore by exposing storage and retrieving methods through a class called *PersistanceManager*. To use this class, the variable or object that is going to be stored, needs to inherit a class called *Persistence* and implement several methods. This gives overhead because the application needs extra methods in every object that is going to be stored. The PersistenceManager should be able to store objects directly without having to implement a set of methods in every of these objects.

Element	Dependencies in the framework
Classes affected	N/A
Est. time to change	One day
Projected positive outcome	Removes the framework's dependencies to the test applications, and reduces the size of the framework-package
Projected negative outcome	None

Comments:
Because the Peer2Me framework is in an early development stage, several test-applications and debug methods are used throughout the framework. Today it is impossible to compile and create a package of

the framework without including these. Removing these dependencies will reduce the footprint[1] of the framework.

Question 4

How can we add functionality and value to the framework?

The framework can never get enough functionality to suit every situation, but implementing all sorts of functionality into the framework would make it to large and make it impossible to use it on several units. There are however a small list of functionality which can be implemented without making the framework grow out of proportions.
Implementing new functionality can be done in two ways; either directly into the framework, or adding it as optional packages. This would be like Sun has done with Java and the number of JSR-packages that are optional to use and not required unless you need special functionality.
A list of suggestions of new functionality follows.

Element	File IO functionality
Est. time to develop	One week
Projected positive outcome	Adds filehandling to the framework, both read and write ability
Projected negative outcome	Increasing the size of the framework if not included as an optional package

Comments:
Including filehandling into the framework would add valuable functionality which can be useful when developing applications that use the mobile phone's memory card or internal memory. A good example of this is the PeerShare application, see Part V. If the filehandling functionality was available during the implementation of PeerShare, the development time would be reduced by roughly 50%.
Including filehandling into the framework will make the framework larger, but only about 8% larger. An option is to make filehandling available as an optional package.

Element	Xml functionality
Est. time to develop	Two weeks
Projected positive outcome	Adds xml handling to the framework
Projected negative outcome	Increasing the size of the framework if not included as an optional package. Can be very memory and cpu demanding

Comments:
Including the ability to deal with xml-files will give more possibilities. As explained in Section 9.4, xml is suitable for storing information in a structured manner. This would be useful when an application needs to store large amounts of structured data or extract data from prevously stored xml-files. A good example of this is the xml-file used in PeerQuiz. This is explained in Section 14.1. If the functionality required to parse and handle xml-files were included in the Peer2Me framework, the time it took to implement PeerQuiz would be reduced by about 15%.
Creating an xml-parser from scratch is both time-consuming and unecessary, and a possibility is to use existing xml-parsers such as kXml mentioned in Section 9.4.4. Creating a wrapper using kXml and exposing suitable methods would be a good solution.

[1] Footprint of a piece of software is the portion of computing resources, typically RAM, CPU time and disc space, that it requires in order to operate.

Element	Node
Est. time to develop	1 hour
Projected positive outcome	Adds extra information to a node
Projected negative outcome	None

Comments:
In today's version of the framework, the information about a node (both master and slave nodes) is limited to nodename, address and a description. It is often required to store extra information about the node, and doing this in todays version involves creating a new object that inherits the node-object. This can easily be avoided by adding a new `object` datatype with *get* and *set*-methods in the node-class. This will only increase the size of the framework by a fraction, and is done in an hour.

22.2 End-user questions

The end-user group consisted of five people; Torbjørn Vatn, Steinar Hestnes, Jørgen Rygh and us. We chose to have a small group so that we could discuss issues and help each other during the testing. We explained how the applications worked and all participants got the chance to try out both the master role and slave role. When testing PeerQuiz, we used a Sony Ericsson P900, Sony Ericsson W800i, two Sony Ericsson K750 and a Siemens S65. P900 and S65 does not support PeerShare, so we had to exclude these mobile phones during the testing of PeerShare.

At the end of the session, the end-users were asked 7 questions. We did not answer these questions, since the answers would have been subjective. Figure 22.1 shows pictures of the group testing out the applications.

22.2.1 Answers to PeerQuiz

1. What kind of functionality does the application lack?
Torbjørn: Would like to know the accumulated score during the quiz.
Steinar: Would like to know the correct answers to all the questions, so that I can learn more.
Jørgen: Would like to download new questions from somewhere.

2. What problems did you experience?
Torbjørn: Had some connection problems.
Steinar: The person that hosted the quiz did not always find all the slaves.
Jørgen: No problems.

3. What are the positive sides of the application?
Torbjørn: Persistent and fun to use.
Steinar: Nice graphical user interface. Easy to use.
Jørgen: Easy to use.

4. What are the negative sides of the application?
Torbjørn: Slightly difficult to use. You have to do things in a certain order. I would also like to know the scores of the other players during the quiz.
Steinar: See answer in 1.
Jørgen: Too few questions in the application. Would like to have more.

5. What do you think is the MOST important factor for this kind of application; Usability, Stability, Performance, Functionality or Entertainment value?
Torbjørn: Usability.

Figure 22.1: *Mobile ad hoc collaboration in a natural environment*

Steinar: Usability.
Jørgen: Entertainment value.

6. What do you think of the usability of the application (1=poor, 10=fantastic) and why?
Torbjørn: 4. Slightly difficult to use. Everybody has to join before the master hosts the quiz. It seems a bit strange.
Steinar: Difficult to answer. Was instructed on how to do things, did not get the chance to find out on my own.
Jørgen: 7. It is easy to use.

7. Do you see yourself using such applications often (daily, monthly, never) and why?
Torbjørn: Never. To hard to write on the mobile phone and all participants must be in close proximity of each other.
Steinar: Probably never.
Jørgen: Never. Would rather play trivial pursuit in "real" life.

22.2.2 Answers to PeerShare

1. What kind of functionality does the application lack?
Torbjørn: Transfer of other files that textfiles.
Steinar: Killer application. The possibility to transfer other files than textfiles would be ultimate.
Jørgen: Notification when a peer is available and the ability to download a folder.

2. What problems did you experience?
Torbjørn: Was not able to create folders on the mobile phone. Had to connect the mobile phone to my laptop in order to create folders with some textfiles.
Steinar: Configuration of shared folders.
Jørgen: Was difficult to set up the folders.

3. What are the positive sides of the application?
Torbjørn: Innovative. Good user interface.
Steinar: Killer application.
Jørgen: Interesting concept, nice timing with the coming of mp3 capable phones.

4. What are the negative sides of the application?
Torbjørn: Slightly difficult to use. You have to do things in a certain order. Only able to transfer textfiles.
Steinar: The usability of sharing folders and specifying download folders.
Jørgen: Unable to transfer binary files.

5. What do you think is the MOST important factor for this kind of application; Usability, Stability, Performance, Functionality or Entertainment value?
Torbjørn: Usability.
Steinar: Usability.
Jørgen: Functionality.

6. What do you think of the usability of the application (1=poor, 10=fantastic) and why?
Torbjørn: 8. Fancy. But a bit hard to create upload and download folders.
Steinar: Difficult to answer. Was instructed on how to do things, did not get the chance to find out on

my own.
Jørgen: 6. Cool design.

7. Do you see yourself using such applications often (daily, monthly, never) and why?
Torbjørn: Monthly, but only if other files than textfiles can be transferred. Would like to transfer music and pictures as well.
Steinar: Weakly or monthly. More useful than the PeerQuiz application.
Jørgen: Probably never. There are easier ways of getting this done.

22.3 Summary

During this evaluation we have found many aspects of the Peer2Me framework that needs improvement. The developer questions have helped us in locating bugs and made us consider how we can improve the framework. The suggested improvements should not be implemented without careful consideration. The end-user testing also proved to be useful. By testing the applications in a natural environment, we discovered more issues and got valuable feedback from our participants. The participants pointed out problems that are directly related the Peer2Me framework, for instance that the framework only supports transfer of ASCII.

A prioritized list of improvements with reasonings could be a start in deciding which improvements should be made. It is also important to pay attention to Sun's development of J2ME and the optional packages while doing this.

CHAPTER 23

Summary of the project

This chapter summarizes what we have learned during this project and discusses the current version of Peer2Me. There are a lot of work that needs to be done to the framework before promoting it to developers. This work is discussed in the last section of this report.

23.1 Conclusion

The objective of this project was to evaluate the Peer2Me framework. We did this by using the engineering method approach as described in Section 3.2 by developing two applications designed to utilize as much of the framework as possible. This was done successfully and we have learned much during this process. The two applications were tested on end-users, which we received constructive feedback from. This feedback along with our own discoveries have been valuable when evaluating the framework.

The Peer2Me framework is a project that looks very promising. There is currently a shortage of wireless mobile ad hoc applications available and Peer2Me is a framework that is designed to make it easier to develop such applications. The development in the market of mobile phones and the increasing capacity of such devices encourage further work with Peer2Me. Wireless mobile ad hoc networks will add value to owners of mobile phones and applications on such networks are and will continue to be developed. Therefore, we highly recommend that the work with Peer2Me continues and hope some day to see the framework widely adopted among developers.

23.2 Further work

The answers to the research questions in Chapter 22 can be used as a guide to further work. Sars and Norum [22] mention some issues that are similar to ours which this section discusses. The suggested further work in this section are presented as short-term and long-term goals

23.2.1 Short-term goals

Fix bugs - We discovered a few bugs in the Peer2Me framework, see Chapter 22, which should be fixed. Some of the bugs require more time to fix than others. All the bugs have been given a priority that

is related to the impacts the bug has on the framework. The bugs should be fixed before adding new features to Peer2Me.

Evaluate suggested new functionality - In addition to the bugs, we have suggested new functionality that the framework might benefit from offering. These suggestions can be found as answers to research question 4 in Chapter 22. However, before implementing these new functionalities, they should be further evaluated. The evaluation should include in-depth analysis of advantages and disadvantages of the suggestions.

Implement new functionality - The outcome of the evaluation should result in some new functionality that can be added to Peer2Me. It is important that these new functionalities follows the existing package structure in Peer2Me. As discussed in Chapter 22 some of the functionality may be offered as optional packages for the Peer2Me framework. Not all applications will need these new features, and putting them in optional packages will reduce the footprint of the framework.

Investigate security issues - In our prestudy we have included some security issues regarding wireless network technology, see Section 9.3. Future ad hoc applications might require stricter security than what is offered by the Peer2Me framework today. Studies can be performed in the area of security in Bluetooth, and can then be used to enhance the security features in Peer2Me. This will enable developers to create applications that transfers sensitive information between mobile phones such calendar information.

23.2.2 Long-term goals

Implement pure P2P - The current version of Peer2Me incorporates a hybrid P2P model due to restrictions in the underlying Bluetooth network technology. If future Bluetooth technology allows a node to function as a master and a slave at the same time, a scatternet network can be created. This will allow Peer2Me to incorporate a pure P2P model. The load will then be balanced on all the nodes in the network and not just the master which is the case today. Another way to implement a pure P2P model, would be to use another wireless network technology such as WLAN, but this requires much more resources from the mobile device than Bluetooth.

Large scale developer test - There has not been conducted a large scale developer test of the Peer2Me framework yet. After a satisfying version of the Peer2Me framework has been developed, such a test would give valuable feedback which is of high interest to the project. The test must be carefully planned and supervised, which is a great challenge.

Release an official version - A milestone in the Peer2Me project is the release of an official version. If the Peer2Me framework is adopted by large community of developers, then Peer2Me can be said to have become a success.

Bibliography

[1] S. E. M. C. AB. p900 - overview. Retrieved October 19th, 2005, from `http://www.sonyericsson.com/spg.jsp?cc=gb&lc=en&ver=4000&template=pp1_loader&php=php1_10101&zone=pp&lm=pp1&pid=10101`., 2005.

[2] S. E. M. C. AB. W800i - overview. Retrieved October 19th, 2005, from `http://www.sonyericsson.com/spg.jsp?cc=gb&lc=en&ver=4000&template=pp1_loader&php=php1_10245&zone=pp&lm=pp1&pid=10245`., 2005.

[3] W.-F. Alliance. Deploying wi-fi protected access (wpa) and wpa2 in the enterprise. 2005.

[4] V. R. Basili. Experimental software engineering issues: Critical assessment and future directions. *The Experimental Paradigm*, 1992.

[5] L. Bass, P. Clements, and R. Kazman. *Software in Practice, Second Edition*. Addison Wesley, 2004.

[6] BEDD. Bedd^{TM}bringing people together. Retrieved November 2th, 2005, from `http://www.bedd.com/about.html`., 2005.

[7] M. Blaser. Industrial-strength security for zigbee: The case for public-key cryptography. 2005.

[8] J. Brucker-Cohen and K. Moriwaki. Umbrella.net. Retrieved Desember 7th, 2005, from `http://www.undertheumbrella.net/`., 2005.

[9] J. Brucker-Cohen, K. Moriwaki, and L. Doyle. Umbrella.net : exploring coincidence ad-hoc networks. Technical report, Networking and Telecommunications Research Group, Trinity College Dublin, 2004.

[10] M. Conti. Body, personal, and local ad hoc wireless networks. In *The handbook of ad hoc wireless networks*, pages 3–24. CRC Press, Inc., Boca Raton, FL, USA, 2003.

[11] C. Ellis, S. Gibbs, and G. Rein. Groupware: Some issues and experiences. *Communcations of the ACM*, 34, January 1991.

[12] G. H. Forman and J. Zahorjan. The challenges of mobile computing. Technical report, University of Washington, 1994.

[13] J. S. Gerd Kortuem, D. Preuitt, T. G. C. Thompson, S. Fickas, and Z. Segall. When peer-to-peer comes face-to-face: Collaborative peer-to-peer computing in mobile ad hoc networks. Technical report, Department of Computer and Information Science, University of Oregon, 2001.

[14] O. S. T. Group. Blue cove. Retrieved Desember 7th, 2005, from `http://sourceforge.net/projects/bluecove/`., 2005.

[15] M. Guenes. Classification of ad-hoc networks. Retrieved October 11th, 2005, from http://www.adhoc-nets.de/., 2004.

[16] B. Hui. Connecting pc and phone with java bluetooth api part 1. Retrieved November 2th, 2005, from http://www.benhui.net/modules.php?name=Bluetooth&page=Connect_PC_Phone_Part_1.html., 2005.

[17] S. M. Inc. Java^{TM}2 platform, micro edition: The java tm platform for consumer and embedded devices. 2002.

[18] S. M. Inc. Java 2 platform, micro edition (j2me). Retrieved November 25th, 2005, from http://java.sun.com/j2me/., 2005.

[19] M. T. Ionita, D. K. Hammer, and H. Obbink. Scenario-based software architecture evaluation methods: An overview. Technical report, Department of Mathematics and Computing Science, XXXX.

[20] kXml. kxml, a simple pull-parser. Retrieved October 21th, 2005, from http://kxml.sourceforge.net/., 2005.

[21] A. Laurie and B. Laurie. Serious flaws in bluetooth security lead to disclosure of personal data. Technical report, A.L. Digital Ltd, 2003.

[22] C.-H. W. Lund and M. S. Norum. The peer2me framework, a framework for mobile collaboration on mobile phones. Master's thesis, NTNU, 2005.

[23] Q. H. Mahmoud. "wireless application programming with j2me and bluetooth". "February" 2003.

[24] N. Maibaum and T. Mundt. Jxta: A technology facilitating mobile peer-to-peer networks. Technical report, University of Rostock; Department of Computer Science, Germany, 2002.

[25] S. Microsystems. Jsr 259: Ad hoc networking api. Retrieved December 8th, 2005, from http://www.jcp.org/en/jsr/detail?id=259., 2005.

[26] S. Microsystems. Jsr 82: Java^{TM}apis for bluetooth. Retrieved December 8th, 2005, from http://www.jcp.org/en/jsr/detail?id=82., 2005.

[27] B. Mobile. s65 - style meets performance. Retrieved October 19th, 2005, from http://www.benqmobile.com/cds/frontdoor/0,2241,hq_en_0_27139_rArNrNrNrN,00.html., 2005.

[28] S. Norway. Retrieved November 19th, 2005, from http://www.ssb.no.

[29] OASIS and R. T. Commitee. Relax ng home page. Retrieved November 19th, 2005, from http://www.relaxng.org/., 2005.

[30] R.E.Johnson and B.Foote. Designing reusable classes. *Journal of Object-oriented Programming*, 1988.

[31] R. R.E.Kraut, R.S.Fish and B.L.Chalfonte. Informal communication in organizations: form, functions and technology. *People's reactions to technology in factories, offices and aerospace*, 1999.

[32] G. Rohan and B. Arun. Bluetooth ad-hoc networking for inter-vehicle communication. Retrieved Desember 7th, 2005, from http://www.undertheumbrella.net/., 2004.

[33] W. Schools. Dtd tutorial. Retrieved November 20th, 2005, from http://www.w3schools.com/dtd/default.asp., 2005.

[34] W. Schools. Introduction to xml schema. Retrieved November 19th, 2005, from http://www.w3schools.com/schema/schema_intro.asp., 2005.

[35] W. Schools. Xml dom tutorial. Retrieved November 19th, 2005, from http://www.w3schools.com/dom/default.asp., 2005.

[36] SIG. The official bluetooth website. Retrieved November 26th, 2005, from `http://www.bluetooth.com/`., 2005.

[37] M. Thoresen. Peer-to-peer systems. Technical report, Institutt for datateknikk og informasjonsvitenskap, 2003.

[38] VeriSign. Code signing for digital ids. Retrieved November 16th, 2005, from `http://www.verisign.com/products-services/security-services/code-signing/digital-ids-code-signing/index.html`., 2005.

[39] Wikipedia. Mobile ad-hoc network. Retrieved October 7th, 2005, from `http://en.wikipedia.org/wiki/Mobile_ad-hoc_network`., 2005.

[40] Wikipedia. Obex. Retrieved November 20th, 2005, from `http://en.wikipedia.org/wiki/OBEX`., 2005.

[41] Wikipedia. Open mobile alliance. Retrieved November 20th, 2005, from `http://en.wikipedia.org/wiki/Open_Mobile_Alliance`., 2005.

[42] Wikipedia. Peer-to-peer. Retrieved September 25th, 2005, from `http://en.wikipedia.org/wiki/P2p`., 2005.

[43] Wikipedia. Personal area network. Retrieved October 7th, 2005, from `http://en.wikipedia.org/wiki/Personal_area_network`., 2005.

[44] Wikipedia. Syncml. Retrieved November 20th, 2005, from `http://en.wikipedia.org/wiki/SyncML`., 2005.

Part VII

Appendix

APPENDIX A

Snapshots of PeerQuiz

Figure A.1: *Splash screen*

Figure A.2: *Main menu*

Figure A.3: *Settings menu*

Figure A.4: *Choosing quiztype*

Figure A.5: *Custom quiz*

Figure A.6: *Entering nickname*

Figure A.7: *Entering nickname*

Figure A.8: *Request to allow Bluetooth connection*

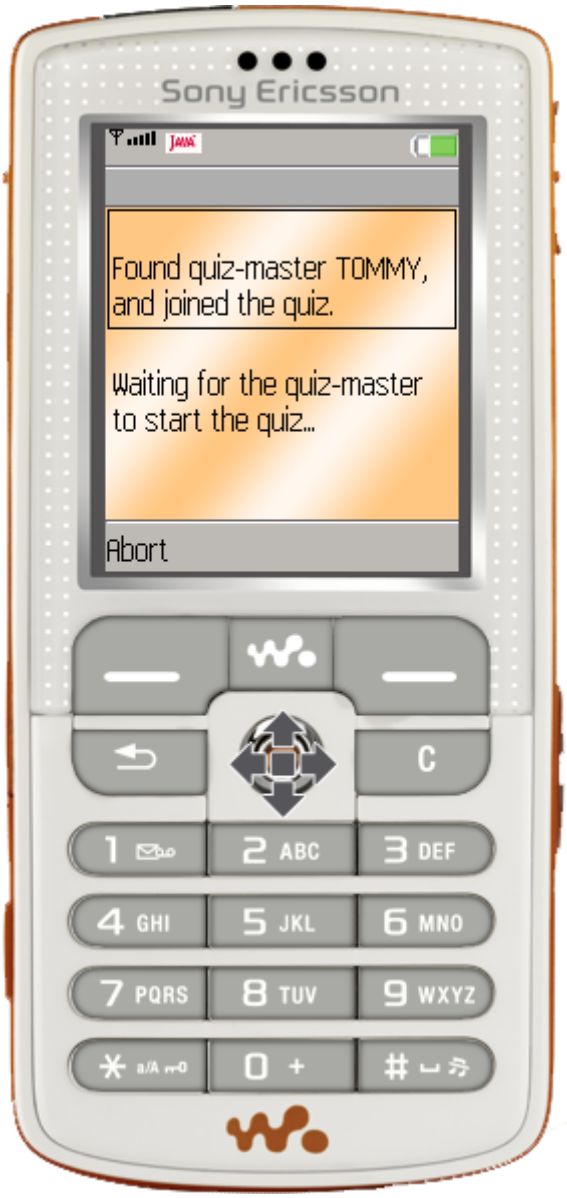


Figure A.9: *Searching for master*

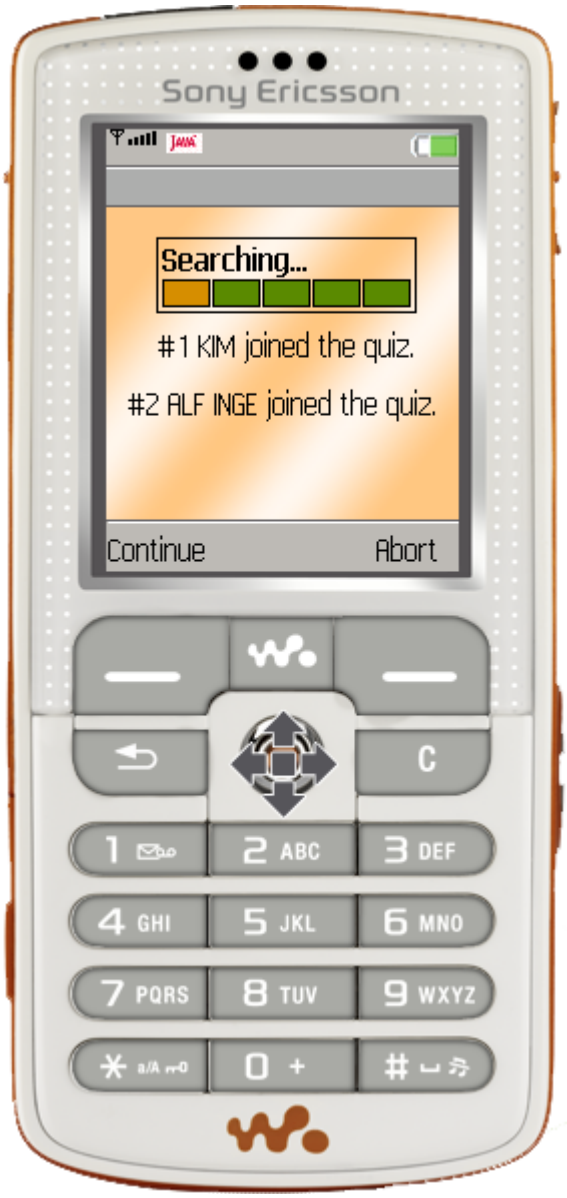


Figure A.10: *Searching for slaves*

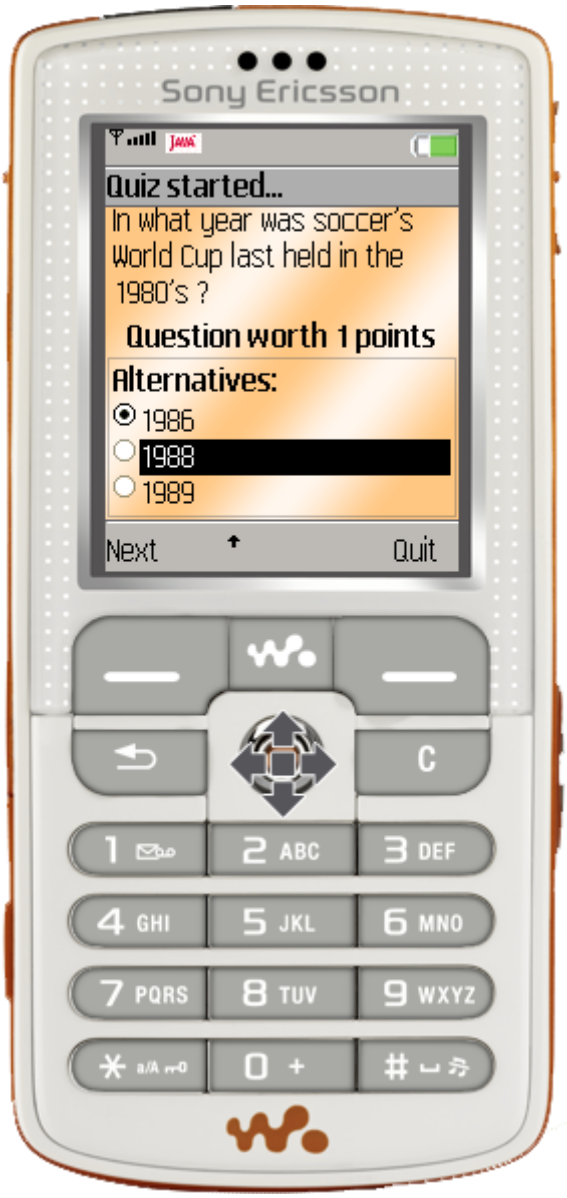


Figure A.11: *Answering questions*

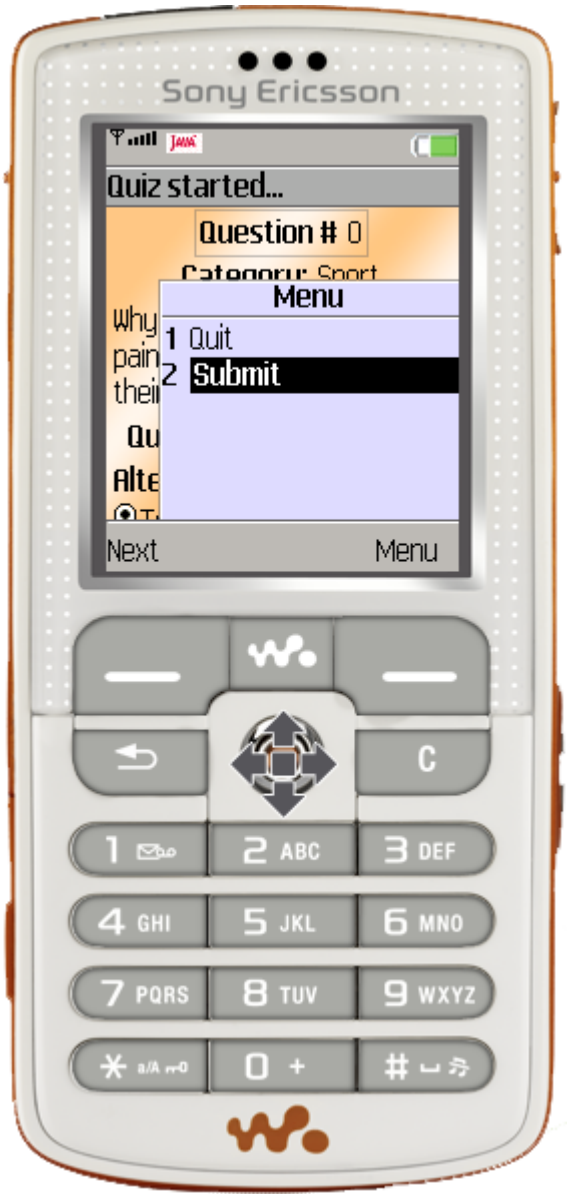


Figure A.12: *Submitting answers*

Figure A.13: *Winner declared*

APPENDIX B

Snapshots of PeerShare

Figure B.1: *Main menu*

Figure B.2: *Settings menu*

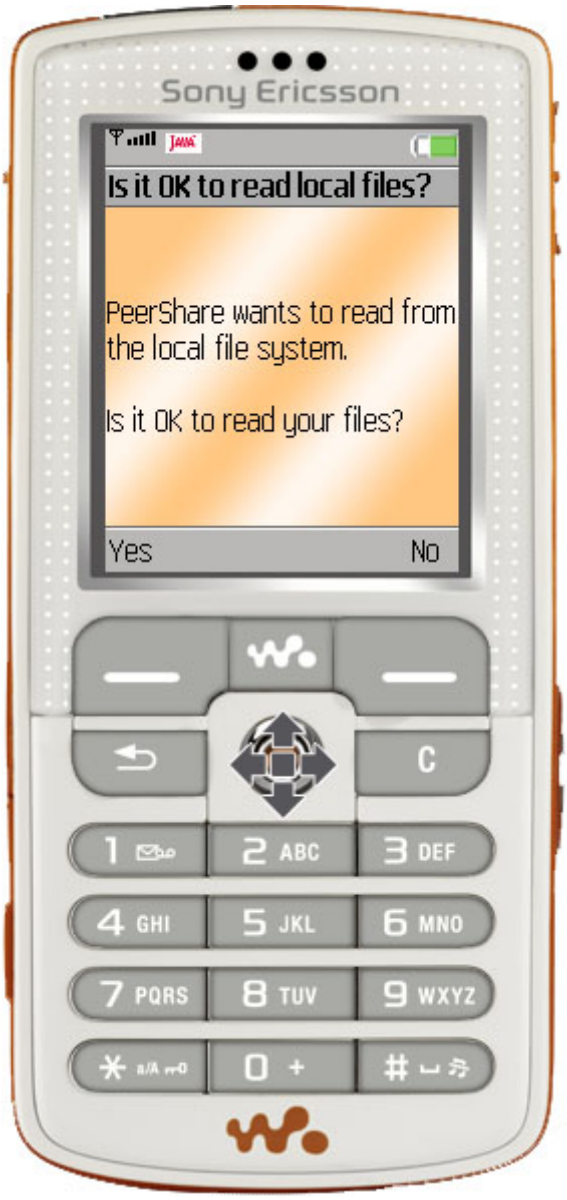


Figure B.3: *Permission to access file system*

Figure B.4: *Selecting download and upload folders*

Figure B.5: *Selecting download folder*

Figure B.6: *Download folder is selected*

Figure B.7: *Download & Upload folders selected*

Figure B.8: *Entering nickname*

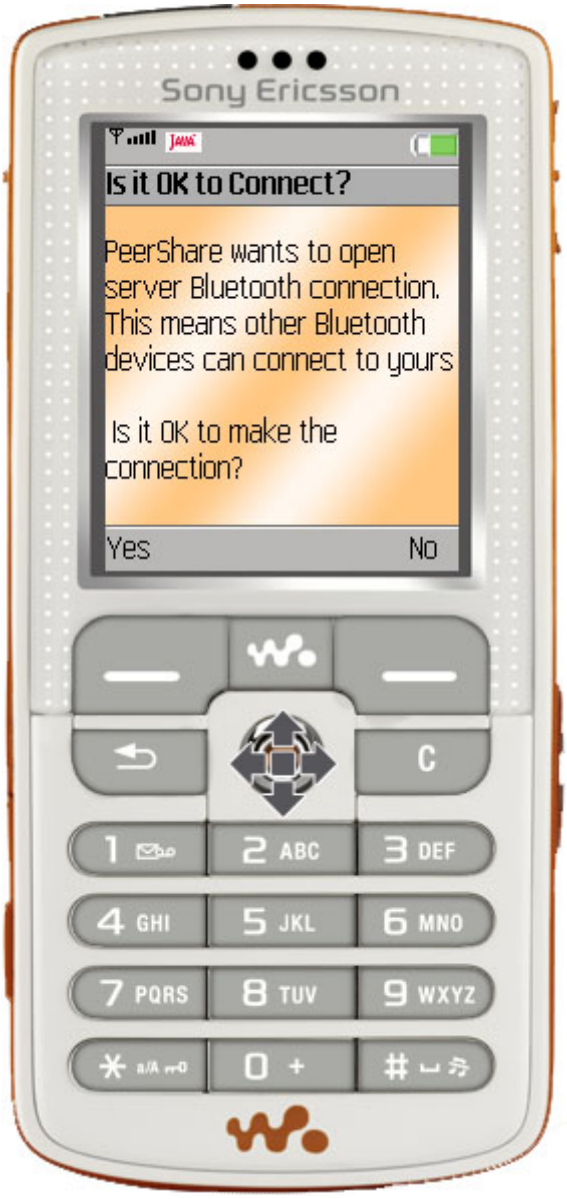


Figure B.9: *Request to allow Bluetooth connection*

Figure B.10: *A master's perspective - Nodes found*

Figure B.11: *A slave's perspective - Nodes found*

Figure B.12: *Browsing the connected peer's files*

APPENDIX C

Questionnaire for Peer2Me developer testing

This is the same questionnaire that Sars and Norum gave to their participants in [22]. Some questions have been removed and some have been slightly modified.

Background and Experiences

1. Describe your programming skills. (Professional, Good, Medium, Poor, None)
Kim: Good
Tommy: Good

2. Describe your Java skills. (Professional, Good, Medium, Poor, None)
Kim: Good
Tommy: Good

3. Have you ever developed a mobile application before?
Kim: No
Tommy: No

4. Have you ever developed using J2ME?
Kim: No
Tommy: No

5. Have you ever developed a Bluetooth application?
Kim: No
Tommy: Yes

6. Do own a mobile phone supporting Bluetooth?
Kim: Yes
Tommy: Yes

The Domain Concepts

7. I think the domain concepts of Peer2Me are easy to understand. (Completely agree, Agree, Neutral, Disagree, Completely disagree)
Kim: Agree
Tommy: Neutral

8. I think the concept Framework is easy to understand. (Completely agree, Agree, Neutral, Disagree, Completely disagree)
Kim: Neutral
Tommy: Agree

9. I think the concept Node is easy to understand. (Completely agree, Agree, Neutral, Disagree, Completely disagree)
Kim: Completely agree
Tommy: Completely agree

10. I think the concept Network is easy to understand. (Completely agree, Agree, Neutral, Disagree, Completely disagree)
Kim: Neutral
Tommy: Neutral

11. I think the concept Service is easy to understand. (Completely agree, Agree, Neutral, Disagree, Completely disagree)
Kim: Agree
Tommy: Disagree

12. I think the concept Group is easy to understand. (Completely agree, Agree, Neutral, Disagree, Completely disagree)
Kim: Agree
Tommy: Completely agree

13. I think the concept Message is easy to understand. (Completely agree, Agree, Neutral, Disagree, Completely disagree)
Kim: Agree
Tommy: Completely agree

14. I think these concepts simplifies the problem domain. (Completely agree, Agree, Neutral, Disagree, Completely disagree)
Kim: Agree
Tommy: Agree

The Peer2Me Development Guide

15. I think the development guide is easy to read. (Completely agree, Agree, Neutral, Disagree, Completely disagree)
Kim: Neutral
Tommy: Disagree

16. I think the development guide helped me through the exercise. (Completely agree, Agree, Neutral, Disagree, Completely disagree)

Kim: Agree
Tommy: Neutral

The development of test-applications

17. What do you think was the most difficult part of using the Peer2Me framework? (Preparing the Framework, Searching for Other Devices, Monitoring the Group, Sending a Message, Receiving a Message, When Something Goes Wrong)
Kim: When Something Goes Wrong
Tommy: Searching for Other Devices

18. What was the most time consuming part regarding the Peer2Me framework? (Preparing the Framework, Searching for Other Devices, Monitoring the Group, Sending a Message, Receiving a Message, When Something Goes Wrong)
Kim: When Something Goes Wrong
Tommy: Preparing the Framework

Summary

19. I think Peer2Me speeds up the development of collaborative mobile applications. (Completely agree, Agree, Neutral, Disagree, Completely disagree)
Kim: Neutral
Tommy: Agree

20. I think Peer2Me clarifies the domain of mobile collaborative applications. (Completely agree, Agree, Neutral, Disagree, Completely disagree)
Kim: Agree
Tommy: Neutral

APPENDIX D

Contents of CD-Rom

Report

- Report in *pdf*-format

PeerQuiz

- Source code
- Jar-package
- Javadoc

PeerShare

- Source code
- Jar-package
- Javadoc