The Peer2Me Framework
A Framework for Mobile Collaboration on Mobile Phones

Carl-Henrik Wolf Lund and Michael Sars Norum

June 2, 2005

Abstract

This project continues the work started in our depth study project in the fall of 2004, developing a framework for mobile collaborative applications on mobile phones utilizing Personal Area Networks (PANs).

This paper describes central, theoretical concepts connected to the Peer-to-Peer (P2P) computing, the Mobile Ad Hoc NETworks (MANETs) and the Computer Supported Cooperative Work (CSCW) domains, focusing on "Same-Place-Same-Time" collaboration. We argue how the spread of PAN technology and mobile phones enable for a broad range of new collaborative applications supporting both collocated work and spontaneous interaction. Updated information about relevant technologies and related projects are discussed and evaluated.

The requirements for the Peer2Me framework are presented and updated along with a revised and improved design. The design and the requirements are a product of an explorative development effort to develop the next generation of the Peer2Me framework using Java 2 Micro Edition and the Java APIs for Bluetooth wireless technology (JABWT). The Peer2Me framework is then tested on actual developers in a workshop arranged in May 2005. Data gathered from this workshop is used to illustrate the benefits of using a framework like Peer2Me for developing mobile collaborative applications.

In addition to the actual Peer2Me framework implementation along with its Bluetooth network module, example applications are designed, implemented and tested in order to verify the suitability of the Peer2Me framework in the problem domain. These applications illustrate different kinds of aspects of the Peer2Me framework and the domain of mobile collaborative applications. The tests of these applications are done through enactment of the usage scenarios from which the applications were derived.

The main results of this project are the technical products comprised of the Peer2Me framework, the Bluetooth Network module and the example Peer2Me applications, as well as the empirical data supporting the advantages of Peer2Me and the evaluations upon the suitability of the applied technologies.

Preface

This master thesis documents the work Carl-Henrik Wolf Lund and Michael Sars Norum have contributed to the Peer2Me project from January to June in 2005. The Peer2Me project is related to the MOWAHS (MObile Work Across Heterogenous Systems) project run by the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU).

Acknowledgements

We would like to thank Alf Inge Wang for his guidance and sharing of expertise during our work on this thesis. His support and advice has been invaluable to us along the way.

We would also like to thank the participant in our developer workshop: Tore Aabakken, Trond Marius Øvstetun, Stein Kåre Skytteren, Øivind Røed, Christain Marshall Rieck, Erik Axel Nielsen and Arne Johan Hestnes.

Trondheim, June 2, 2005.

Carl-Henrik Wolf Lund

Michael Sars Norum

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the last decade mobile phones have become a major part of our every day life. In addition, the tasks we solve daily have grown more complex thus making human collaboration more complex, increasing the benefit we have from using computers to support interaction. Traditional computers are much too large and heavy to be carried around at all times and used for spontaneous collaboration, but mobile phones have become a very viable option.

Mobile phones today have approximately 96% coverage among the population in Norway, Sweden follows close with around 90% of the population using a mobile phone, [43]. This high spread of mobile phones makes them an untapped resource for deploying applications with the potential of achieving a large user base.

Today's mobile phones soon have almost the same computational capacity as the personal computers had when the research field of Computer Supported Cooperative Work (CSCW) first emerged. In addition to becoming quite computationally powerful, mobile phones also have the advantage of being small and almost always on, always present and connected. The evolution of these devices to the state they are in today, have enabled us to focus CSCW research on collaboration in its true form, mobile collaboration. Mobile CSCW has numerous advantages compared to traditional CSCW using stationary devices, since human interaction seldom is completely stationary. We move around the office, the city, the country or even the world in our everyday interaction with other persons in our environment, mobile CSCW gives us the ability to have computer support anywhere, anytime.

The emergence of mobile phones with some kind of ad hoc network technology built-in creates opportunities for new forms of computerized collaboration. Mobile and computer supported collaboration between people that are collocated is now possible. This project will aim to design and implement a framework used to create such collaborative applications on mobile phones. The name of the framework is Peer2Me, which is an abbreviation for Peer-to-peer (P2P) for Java 2 Micro Edition (J2ME).

1.1 Motivation

Personal Area Networks (PANs) are low cost, low range networks that allow users to create spontaneous ad hoc networks that do not depend on any central node. These ad hoc network technolo-

gies enable devices to detect and connect to other devices that are in sufficient proximity and form mobile P2P networks. P2P networks is a special kind of networks where all nodes have equal functionality. Ad hoc network technologies are ideal for transmitting information between devices that are physically collocated. With PANs our mobile phones can participate with us in face-to-face interaction. Such technologies as for instance Bluetooth, InfraRed (IR) or WLAN are more and more widely supported on mobile phones. Utilizing PAN technology for mobile phones enables for a broad range of new categories of collaborative applications supporting collocated work and spontaneous interaction. Different kinds of application scenarios has been previous described by Kortuem in [36] and by Heinemann in [25].

Previously, many research projects in the field of mobile collaboration have produced specialized prototypes of mobile P2P and collaborative applications to demonstrate the potential of the emerging technologies. Developing networked applications requires a lot of effort from developers and researchers when it comes to understanding the underlying technology, establishing network infrastructures and designing communication protocols. The researchers spend a vast amount of time developing these prototypes because everyone is starting from scratch, developing their own unique architectures and designs. No one has yet succeeded in making a general framework for these types of applications on mobile phones.

When we use the term framework we refer to an object oriented framework, defined in [31] as:

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

In general, instead of designing a specific application architecture, a developer might choose to develop a framework. The framework will allow the developer to generate a collection of applications for an entire domain. It might require a significant larger amount of time to develop a framework rather than a single application. But a completed framework enables higher productivity and shorter development time for the actual applications, because of its capability of code and design reuse.

The mobile application market is moving fast, new, more powerful phones are introduced and the average phone has a lifetime before renewal of approximately two years. This short renewal time speeds up the adoption of new technology and creates a demand for applications utilizing this technology. As stated by Tétard in [56], if a new application has a too long time to market, it will most probably be out of date by the time it is finished. The task of creating the infrastructure for a network can be a quite tedious effort, increasing the applications time to market. By creating a framework for applications using ad hoc networks we will introduce a common building block to build numerous applications on. The applications' time to market can be reduced drastically if such buildings block are available to developers.

According to Gert Kortuem, et al. in [36], experiences from the Proem projects shows that students in an advanced system engineering course were able to develop complex P2P applications relatively fast by using the Proem framework. This clearly states that a substancial benefit can be gained from having an ad hoc networking framework available when developing applications.

On November 2nd 2004, the Java Community Process (JCP) issued a Java Specification Request (JSR), number 259. The JSR 259 aims to specify an Application Programming Interface (API) for Ad Hoc Networks for Java 2 Micro Edition (J2ME), [4]. Major mobile phone vendors such

as Siemens, Nokia and Panasonic initiated this project and its goals overlap with the Peer2Me framework. The issuing of the JSR 259 illustrates the relevance of creating a framework for ad hoc networking on mobile devices with J2ME support. More information about the JSR 259 can be found in Chapter 5.2. There have been numerous attempts at creating a framework or API for similar purposes before, all of which have failed. At the end of a project conducted during the fall of 2004, we developed a prototype framework in J2ME supporting basic functionality needed by collaborative applications on mobile phones. In addition we made a Bluetooth network module to use as a network layer in the framework. Although this framework was not in perfect working condition, it served its purpose as a proof of concept showing how developers can benefit from using such a framework.

By having a framework that deals with the network infrastructure and handles the communication between all nodes in the network and provides an application's interface, we think the development time for collaborative mobile applications can be reduced. This will be very useful for us and other researches when making proof-of-concept applications. A well documented framework will also establish terms and definitions that can be used by researchers and other participants in the research field. A framework for developing applications will also hopefully reduce the time before someone creates a killer application that will help the field of mobile collaboration to gain acceptance in the consumer market. All in all we would like to contribute to the development of a standard for ad hoc collaborative applications on mobile phones.

1.2 Problem Definition

Currently there are no available APIs or frameworks for writing applications that utilize mobile phones and personal area ad hoc networks. Since mobile applications need to be developed fast and because developing such applications requires a lot of effort, such a framework is needed. Java 2 Micro Edition and Bluetooth are currently available on a number of mobile phones and is ideal for testing the theories related to the creation of such a framework.

The aim of this master thesis is to design, implement and test a framework for mobile collaborative applications using Java 2 Micro Edition and Bluetooth. The design and implementation will be based upon a prototype framework developed during our depth project described in [37]. The testing of the framework will be conducted by developing example applications based on different usage scenarios. It should also be conducted some empirical work to test the usefulness and usability of the framework for developers.

In addition to the development of the framework and the writing of this report, the work should also be published on the Peer2Me website[1]. Enabling other developers to download and use the Peer2Me framework in their own applications and hopefully provide feedback to us on their experiences.

1.3 Limitations of Scope

This project's main goal is to evolve and refine the prototype implementation of the framework described in [37]. This gives us a very technical focus throughout the entire project. Our research

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

is placed within the CSCW field, but due to the technical focus we will try to limit our discussions on CSCW issues to a minimum. We will only cover enough of the CSCW field to make our design and implementation choices.

When it comes to more technical limitations, this project will not consider scatternet implementations on Bluetooth. We have found that today's Bluetooth enabled mobile phones are incapable of setting up scatternets. Because of this, we will limit our Bluetooth implementation to cover only piconets. This means for instance that advanced routing or resource allocation algorithms are not needed. A more in depth explanation of these concepts and the differences between them are given in Chapter 3.2.

1.4 Project Context

This project is connected to the MObile Work Across Heterogeneous Systems (MOWAHS) project. MOWAHS is cooperative effort between the Software Engineering and the Database Technology groups of The Department of Computer and Information Science (IDI) at The Norwegian University of Science and Technology (NTNU).

The MOWAHS goals, as stated on the MOWAHS website[2] are:

1. Helping to understand and to continuously assess and improve workprocesses in virtual organizations.
2. 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).
3. Disseminating the results to colleagues, students, companies, and the community at large.

This master thesis is most strongly related to the second goal, creating a development framework and applications to be used in mobile and spontaneous collaborative environments. This report along with the Peer2Me website[3] is connected to the third goal, spreading the results to colleagues, students and the community at large.

1.5 Reader's Guide

The contents of this report vary a lot, both with respect to focus and approach. In order to increase the readability of the report we will now present a short summary of each chapter. In addition to this we would like to list chapters we think may be most interesting to certain stereotypical readers:

Developers interested in writing applications: Should read Chapter 14 and Appendix B. The chapters in Part III can be used as support literature and examples.

[2] `http://www.mowahs.com`
[3] `http://www.peer2me.org`

Readers interested in the problem domain: Should read Chapters 1 and Part I. Some readers may find Part VI useful as well.

Developers wanting to improve Peer2Me: Should read and understand all of Part II. Part I and Part III should be used as support literature in order to understand all central concepts.

Chapter 1 Introduction This chapter describes the motivation behind the project and defines the problem definition.

Chapter 2 Research Questions and Method This chapter describes the research questions that has arised from the problem definition and the motivation. These questions set the focus of the whole project. The different research methods used throughout the project is also described in this chapter.

Part I Prestudy This part of the report documents our project's prestudy.

- **Chapter 3 Central Concepts** This chapter deals with the central concepts related to our problem domain. These concepts should be understood to be able to understand the background and assumptions behind the framework. Most of the following chapters build on the contents of this chapter.
- **Chapter 4 Previous Work** This chapter summarizes the report written during our fall project, [37].
- **Chapter 5 State of the Art** This chapter describes similar or overlapping projects from which we draw experience or knowledge about our problem domain.
- **Chapter 6 Technology** This chapter updates the technology study performed in [37].

Part II The Peer2Me Framework This part of the report documents the requirements, design and implementation of the actual Peer2Me framework.

- **Chapter 7 Requirements** This chapter describes all requirements of the Peer2Me framework, including the requirements gathered in our previous project and the new requirements that have been added during this project.
- **Chapter 8 Design** This chapter gives a complete overview of the design of the framework. Some central concepts of Peer2Me are introduced that the design is based upon. Then an architectural overview is given followed by a detailed design also documenting the changes from the original prototype design. The design of the Bluetooth network module is also presented, illustrating how a network design is created and functioning as a guide for developers wanting to add a new network technology package to the software suite.
- **Chapter 9 Implementation** In this chapter we describe which requirements that are covered by our implementation and how these requirements are covered by the design and implementation. Statistics related to the actual code is given. Code examples of some important implementation details are provided and explained.

Part III The Peer2Me Applications This part of the report documents the example applications we have developed by using the Peer2Me framework.

Chapter 10 Overview This chapter introduces the example applications and provides an overview of how the applications are designed, implemented and tested.

Chapter 11 Business Card Exchange This chapter describes the requirements, design and implementation of the Business Card Exchange application.

Chapter 12 PAN Instant Messaging This chapter describes the requirements, design and implementation of the PAN Instant Messaging application.

Chapter 13 Converging Top Ten List This chapter describes the requirements, design and implementation of the Converging Top Ten List application.

Part IV Developing Peer2Me Applications This part is mainly for developers who want to learn how to use Peer2Me to develop applications, but it can also be read to learn how the different parts of the framework are used by the applications in practice.

Chapter 14 Peer2Me Development Guide This is the main chapter of this part. The chapter provides a walk-through on how to write applications using the Peer2Me framework. Two small example applications are used to illustrate the different aspects of Peer2Me development.

Chapter 11 Using the Persistence Layer in Peer2Me This chapter gives an overview of how to develop applications that use the persistence layer in Peer2Me.

Part V Testing This part of the report documents the testing of the Peer2Me framework.

Chapter 16 Scenario Testing This chapter describes scenario testing of the three scenarios that the example Peer2Me applications are based upon.

Chapter 17 Developer Testing This chapter describes the empirical work that has been conducted to examine to what degree the Peer2Me framework is useful for developers. The results and conclusions drawn from a developer workshop are provided.

Part VI Discussion This part of the report documents and discusses the results and conclusions from this project.

Chapter 18 Encountered Problems This chapter discusses the problems encountered during the lifetime of the project.

Chapter 19 Evaluation This chapter gives an overall technical evaluation of the framework, an evaluation of the empirical work and an evaluation of the three example applications that have been developed.

Chapter 20 Conclusion This chapter describes our conclusions.

Chapter 21 Further Work This chapter describes some of the areas we see as possible future work related to the Peer2Me framework.

Part VII Appendix Appendix A Discovery Time Statistics Statistics of Bluetooth discovery and connection times gathered during scenario testing.

Appendix B The Peer2Me Developer Exercise The exercise used during the developer workshop.

Appendix E Dictionary A list describing the most important terms and abbreviations used in this report.

Appendix C Code Examples The source code for the code examples used in the development guide.

Appendix D Questionnaire for Peer2Me developer testing The questionnaire handed out to the participants of the workshop. Used to evaluate the framework and the workshop and to retrieve data from the participants.

Chapter 2

Research Questions and Method

In this chapter we will describe which questions we seek to answer through our work, and how we intend to find the answers. We will describe how we are planning to conduct the work and explain the different methodologies we are planning to follow.

2.1 Research Questions

The process of creating a framework often includes major challenges and trade-offs. A framework should be flexible and general enough to include solutions for all the different kind of usage scenarios in the problem domain, but still be simple and usable enough to enable high productivity. To what degree will developers benefit from using such a framework for developing? The domain of mobile collaborative applications are quite large, involving totally different usage scenarios, so its natural to wonder if it is possible at all to create such a framework. The framework has to be based upon new and immature technology. Is this technology suitable and mature enough to be functional for this purpose? A lot of questions have arised from the problem definition and motivation part, and we will now formalize these questions and thoughts into research questions. These research questions will help us focusing our work throughout the project and this report will lead to the answers of these questions.

In this project we want to find the answers to the following research questions:

1. Is it technical possible to develop and implement a framework like Peer2Me for mobile phones?

   - This question will be answered by trying to create the framework itself. This will be done by trying to design, implement and test the framework according to the requirements from [37] and the new requirements that will evolve and occur during this project.

   (a) Are mobile phones together with J2ME a suitable technical platform for mobile collaborative applications?

- This question will be answered by developing the framework itself, developing example applications utilizing the framework and perform scenario testing on these applications.

(b) Is Bluetooth a suitable technology for mobile collaborative applications?

- This question will be answered by implementing and using Bluetooth as an example network module to the framework. By scenario testing example applications utilizing the framework and thereby also the Bluetooth module, it will be possible to evaluate how suitable the Bluetooth technology is for these kinds of applications.

2. Will developers benefit from using Peer2Me when developing mobile collaborative applications on mobile phones?

   (a) Will it be easier to develop applications using Peer2Me?

   (b) Will the development time be reduced when using Peer2Me?

   - These questions will be answered by empirical experiments and qualitative evaluations.

3. Is Peer2Me suitable for developing all categories of mobile collaborative applications described in the classification matrix? These categories of applications are applications that require user interaction (User), automatic collaboration between devices (Auto) and a combination of these (Hybrid). Read more about these categories and the classification matrix in the description of our previous work in Chapter 4.1.1.

4. Are these kinds of collaborative applications useful for end users?

   - These two last questions will both be answered by trying to develop example applications from all the three main categories of applications from the classification matrix. These applications will be tested according to their real-life scenarios to test upon the non functional requirements and to evaluate the usability and usefulness for end users.

We have now defined the research questions for this project and explained shortly how we are going to answer them. In the next section we will describe the different methods, that we are going to use to answer these questions, more in detail.

2.2 Research Method

Software Engineering is a multi disciplined subject. It involves both technical as well as social factors and is therefore a very complex field. Most problems adressed in software engineering are wicked problems, problems with more than one solution where none of the solutions can be selected as the correct solution. When doing research in the software engineering field a formal approach is needed to add credibility to the results. In [11], Basili identifies 3 different approaches to software engineering research. The engineering approach and the empirical approach are part of the scientific paradigm, while the mathematical approach is part of the analytical paradigm. We will now give a summary of each of the three approaches:

The engineering approach: When using the engineering approach, the problem domain is analyzed with respect to existing solutions and then new and improved solutions are proposed. The solutions are built, analyzed and tested until no further improvements can be found.

This approach is very evolutionary and focuses on finding improvements. In order for the approach to be applicable, one has to have a model of the persons, product, process or environment.

The empirical approach: When using the empirical approach, a model of the domain is proposed. From this model statistical/qualitative methods are developed and applied to case studies and analyzed to validate the proposed model.

The empirical approach yields more reliable results since the model must be validated through case studies. Developing a software suite or a software tool alone is not enough to give solve the task.

The mathematical approach: When using the mathematical approach, a formal theory or a set of axioms are proposed. From this, results are derived and if possible compared with empirical data.

For this project we use the engineering approach for continuing the development of the prototype described in [37]. The improved prototype will then be tested for usability and usefulness through the empirical approach. We will now give a more in depth description of the two distinct approaches.

2.2.1 The Engineering Approach

For the actual development of the Peer2Me framework we will use the engineering approach. Our prestudy will be used for analyzing our problem domain with the goal of finding all existing solutions. We will then create an improved solution using formal techniques for finding requirements and then use a iterative development cycle to refine our solution as much as possible. We will now describe each of the phases used in our engineering approach more thoroughly.

Prestudy

In [37], we did a thorough study of related and relevant projects as well as a technology evaluation with respect to devices and implementation platforms. The prestudy in this report will be focused on updating the information gathered in [37]. We will give a domain description with central concepts, evaluate any new projects we find that relate to Peer2Me and do a thorough technology update.

Requirements Engineering

In [37], we used a scenario based requirements engineering technique proposed by Sutcliffe in [54]. We found that scenario based requirements engineering based on textual scenarios worked

well for us when elaborating the requirements for the Peer2Me framework. Because of the previous positive experience with this technique, we have decided to use it for finding requirements also for the Peer2Me applications.

Sutcliffe's technique is based on doing a structured analysis of textual usage scenarios. The structured analysis is comprised of the steps shown below:

Goal analysis: The goal analysis starts by looking at the scenario goals. For each goal, the following questions are asked:

1. Does the user's goal require computerized support?
2. Does the goal describe a quality or performance property?
3. If the goal does not require computerized support, can it be achieved by a manual effort?
4. Does the goal require a management decision about resources and responsibilities?
5. Can the scenario goal and its associated task be fully automated?

Inbound event analysis: This step aims to identify the events that take place during a scenario and elaborate these to find their corresponding functional requirements. When all events that have implications for the framework have been uncovered, we will compare these to the requirements found during the goal analysis. If the requirements do not support the event, requirement will be added to cover the event.

Categorize system output: This step aims to identify what output events that are described by the scenarios. Output events are events that can be placed in one of the following categories:

Direct commands: These are events that generate a message to the users requiring human interaction.

Indirect commands: These are events that generate warning messages that recommend human interaction.

Input requests: These are events that need input from the user. Input requests can be considered as a special case of direct commands.

Information displays: The messages generated by these events are strictly informational.

By using this method, we are able to move from a textual description of the applications to a solid set of requirements for each application.

Development Life Cycle

When developing the Peer2Me prototype in the fall of 2004, we used a quite explorative development model described in [37]. This explorative model worked out well for us, enabling us to test and explore the features in J2ME and Bluetooth we were unfamiliar with. Because of our previous success with explorative development, we have chosen to use the same iterative form of working on this master thesis as well. However, we now focus more on developing real applications that

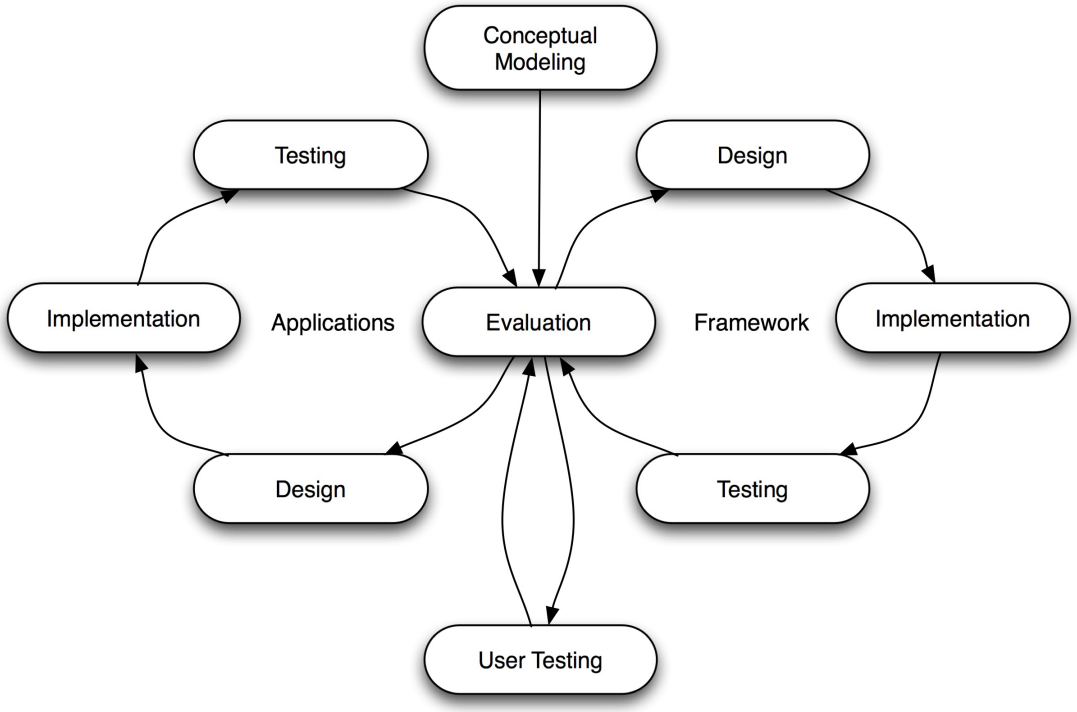


Figure 2.1: Development life cycle, showing the iterative model and the interaction between the application development and the framework development.

utilize the framework than we did in [37]. This shifted focus have forced us to split the development of applications and framework into two different but parallel runs that are kept synchronized at the evaluation phase as shown in Figure 2.1.

The goal of keeping the development processes synchronized is that the application development process probably will uncover weaknesses and shortcomings in the Peer2Me framework itself. By developing the applications in parallel with the framework, we will be able to test the new Peer2Me features as they emerge and quickly discover, implement and test additional functionality required by the applications.

2.2.2 The Empirical Approach

Some of the research questions listed above aim to measure qualitative aspects of the Peer2Me framework. In order to measure these aspects, we need to perform empirical experiments to be able to draw conclusions. Some of them require us to establish formal metrics that can give a quantitative answer to our questions.

One of the goals of this master thesis is to check whether a substancial benefit can be achieved by developing collaborative applications based on a framework providing an API for ad hoc networking. Will the existence and usage of such a framework actually decrease the application's development time and cut the time to market? Along with qualitative evaluation and conclusions from related research we will have to perform real-life developer testing to answer these question.

We also want to do empirical experiments that aims to answer if applications developed with Peer2Me are useful for end users. We will do scenario testing of one application from each of the three categories of applications we proposed in [37], user-triggered applications, applications utilizing automatic collaboration between phones, and a hybrid of these two categories (see Chapter 4.1.1).

Developer testing

During this project we will perform developer testing of our framework. A major testing activity is our own development of Peer2Me applications, documented in Part Three of this report. Further, we also want to test the framework on other developers to get an objective review and to gather evaluation data.

The main goal of the Peer2Me framework is to make it easier for developers to make mobile collaborative ad-hoc applications. To evaluate whether Peer2Me fulfills this goal, we will use both qualitative and quantitative evaluation techniques.

To be able to do a quantitative evaluation, we have to introduce some metrics. You can not understand what you can not measure, the famous physician Michael Faraday once said. To get a better overview of how Peer2Me affects developers, we want to do some experiments to gather some empirical data. The success of empirical experiments is determined by choosing the most appropriate metrics. You need to know what data to gather before you gather it. We will use the Goal Question Metric(GQM) method to break down the goals of Peer2Me to a set of metrics to use in our empirical experiment.

The GQM method was originally developed by V.Basili and D.Weis [61]. The method is based upon both academic research and practical experiences. The basic idea of the GQM method is to derive measurement metrics from measurement questions and goals. The GQM method is widely used within the field of software process improvement based upon the fact that successful improvement of software development is impossible without knowing what your improvement goals are or how to find metrics related to them. The GQM method forces scientists to decide upon and define what they actually want to measure before doing the measuring. It helps out with structuring information associated with measurement.

The GQM method defines a measurement model on three levels [41]:

Conceptual level (goal): A goal is defined for an object, for a variety of reasons, with respect to various models of quality, from various points of view, and relative to a particular environment.

Operational level (question): A set of questions is used to define models of the object of study and then focuses on that object to characterize the assessment or achievement of a specific goal.

Quantitative level (metric): A set of metrics, based on the models, are associated with every question in order to answer it in a measurable way.

A goal is often defined according to a special template called the Goal definition template [63]. This template exists to ensure that all important aspects of the goal are defined in the goal description. The goal template is:

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

The GQM method suggests four different phases in a measurement project [49]:

The Planning phase: The measurement object is selected, defined and characterised and the measurements activities are planned resulting in a project plan.

The Definition phase: The measurement program is defined. The goal, questions, metrics and hypotheses are defined and documented.

The Data collection phase: The actual data collection takes place.

The Interpretation phase: The collected data is processed with respect to the defined metrics. Then these results are used to answer the defined questions. These answers will then be used to evaluate if the planned goal has been reached.

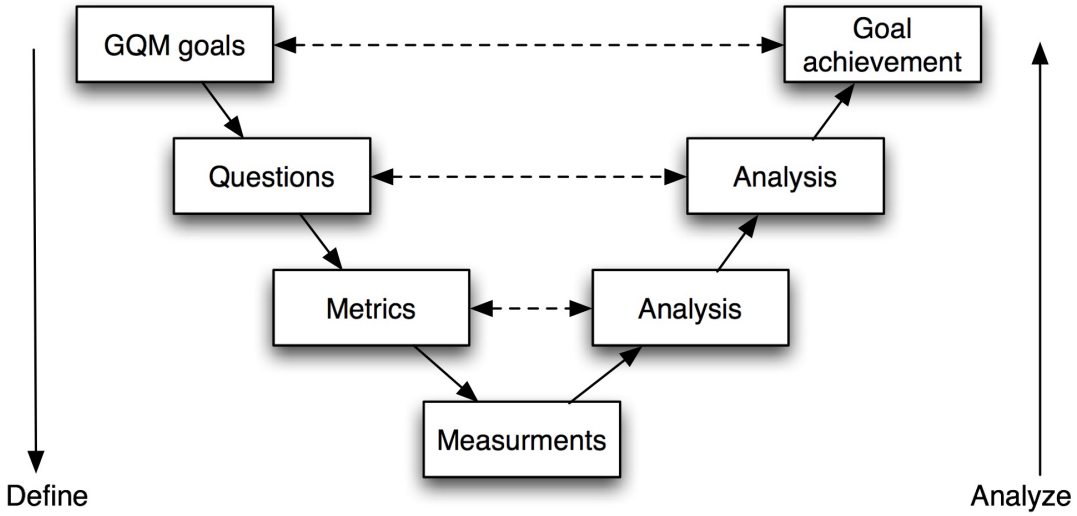


Figure 2.2: GQM V-model.

In the planning phase, goals are defined and then questions and metrics are derived from these goals. In the interpretation phase metrics and questions are used to evaluate upon the goals. According to [19], this can be viewed as a traditional V-model, as seen in Figure 2.2.

We will do a quasi experiment to gather empirical data regarding the usefulness and usability of our framework. This experiment will be realized as a programming exercise done by a group of developers with different backgrounds and experiences. We will use this session to observe how easily developers learn and use Peer2Me. This event will be our data collection phase of our measurement project. To be prepared for this experiment we have analyzed what questions we want to answer. By using the GQM method, we have created GQM trees showing the goals of the experiment as the roots of the trees. From the goals we have derived questions and metrics to answer these questions. The goals are formulated by using the goal definition template given above. The experiment will not provide us with an absolute answer on Peer2Me's usefulness and usability, but will give us a strong and valid indication on it.

GOAL 1: *Analyze* Peer2Me
for the purpose of Deciding if it will be easier to develop applications using the framework
with respect to its Usability, usefulness and effectiveness
from the point of view of the Developers
in the context of Mobile collaborative application development.

QUESTION 1: Is it easy to understand the concepts of Peer2Me?

Metric: Time spent on Peer2Me training.

QUESTION 2: Is it easy to understand the concepts of a Personal Area Network technology as Bluetooth?

Metric: Time spent on learning the concepts of Bluetooth.

QUESTION 3: Is it easy to learn developing mobile collaborative applications with Peer2Me?

Metric: Time spent before mastering Peer2Me development.

Metric: Number of people that succeeds doing a programming exercise.

QUESTION 4: Is it easy to learn developing mobile collaborative applications with Bluetooth?

Metric: Time spent before mastering Bluetooth development.

GOAL 2: *Analyze* Peer2Me
for the purpose of Deciding if it requires less effort to develop applications using the framework
with respect to its Effectiveness and usefulness
from the point of view of the Developers
in the context of Mobile collaborative application development.

QUESTION 1: How much time does it take to develop an application with using Peer2Me?

Metric: Time spent developing

QUESTION 2: How much time does it take to develop an application without using Peer2Me?

Metric: Time spent developing

QUESTION 3: How much source code is produced when developing an application using Peer2Me?

Metric: Lines of code

QUESTION 4: How much source code is produced when developing an application without using Peer2Me?

Metric: Lines of code

The metrics regarding Bluetooth development will not be gathered in the experiment. These values will have to be estimated. It would have been ideal to have an experiment also for gathering these metrics, but it would be way to time consuming. It is also very difficult to find people with Bluetooth programming experience. Instead we will do qualitative analysis based upon our own experiences and knowledge related to Bluetooth programming and mobile ad hoc development in general to draw the conclusions. We will then compare these qualitative data against the data gathered in the experiment.

Scenario Testing

Our research questions asked whether the underlying technologies and the Peer2Me framework itself are suitable for creating applications in each of the three categorizes of mobile collaborative applications (User, Auto and Hybrid). In order to test this we will test the applications in surroundings similar to the scenario environments. These scenario tests will then give the answer to the following parts of our research questions:

- Are mobile phones together with J2ME a suitable technical platform for mobile collaborative applications?

Phone:	Operating System:	CPU:	Java virtual processor:	Java heap size:	Storage Capacity:	Display Size:
Nokia 6600	Symbian OS v7.0	32-bit RISC 104 MHz	35.8 MHz	8977 KB	6 MB	176x208 pixels, 65,536 colors
Sony Ericsson p900	Symbian OS v7.0	32-bit RISC ARM9	54 MHz	4645 KB	16 MB	208x320 pixels, 65,536 colors
Siemens s65	Proprietary Siemens OS	Unknown	33.4 MHz	1500 KB	10.3 MB	132x176 pixels, 65,536 colors

Table 2.1: Mobile phone used for testing and their properties.

- Is Bluetooth a suitable technology for mobile collaborative applications?
- Is Peer2Me suitable for developing all categories of mobile collaborative applications described in the classification matrix?
- Are these kinds of collaborative applications useful for end users?

The focus of the tests will not be to test technical details or functional requirements of the applications, but to verify that the chosen technology and Peer2Me works in the selected scenarios. The tests will be conducted by enacting the scenarios in a manner as close to their original description as possible. A more extensive description of the scenario tests are given in Chapter 16.

2.3 Test Environment

When testing the Peer2Me framework and the applications, our test environment are going to be twofold. Continuous testing of functionality and debugging should be done through the use of the emulator included in Sun Microsystem's Wireless Toolkit version 2.2. In order to make sure that the Peer2Me framework and applications run on actual mobile phones as well, we are going to perform deployment and testing on three different brands of mobile phones. The three phones are a Sony Ericsson p900, a Siemens s65 and a Nokia 6600. All three phones have support for J2ME and the Java APIs for Bluetooth Wireless Technology (JABWT). Table 2.1 shows each phone's properties. The general information is taken from the specifications of the phones and the more specific java properties are taken from a benchmark performed by Club Java [14].

Part I

Prestudy

Chapter 3

Central Concepts

The Peer2Me Framework has arised by combining different concepts from several different research domains. In this chapter we will provide an overview of the most important and relevant aspects of these concepts. The overview will focus on defining the concepts, describing advantages and disadvantages, and summarizing the challenges in the different research domains. By doing this we document and make explicit which design principles, requirements, considerations and technologies that underlies the framework. We will also show how other research domains are related to the framework and how the concepts from these domains influence Peer2Me.

Figure 3.1 shows the central concepts related to this project. Peer-to-Peer (P2P) computing, Mobile Ad Hoc Networks (MANETs) and Computer Supported Cooperative Work (CSCW) all lay down the foundation for this project. Peer2Me supports mobile P2P applications that support collocated work within a Personal Area Network (PAN) with piconet communication only. To understand why and how we have done this focusing and to understand the constraints that are laid upon the Peer2Me design, we are now going to describe these concepts and how they affect and relate to Peer2Me.

3.1 Peer-to-peer Computing

Peer2Me is a framework for developing P2P applications on mobile phones. P2P computing is an alternative architecture to the centralized models of computing. According to Dejan S. Milojicic et al., computer systems can be classified into centralized and distributed [40] (see Figure 3.2). Distributed systems can further be classified into a client-server model and a P2P model. In a client-server model, the server is the central entity and the only provider of service and content. In P2P computing, resources are shared between peers operating as both servers and clients. The P2P model can be either pure or hybrid.

The term *peer* stems from Latin for equal, and to understand what P2P computing is, we first have to define this term. A peer is a node in a P2P network. It is the fundamental processing unit of any P2P solution. Brendon J. Wilson defines a peer in [62] as:

> "Any entity capable of performing some useful work and communicating the results of that work to another entity over a network, either directly or indirectly."

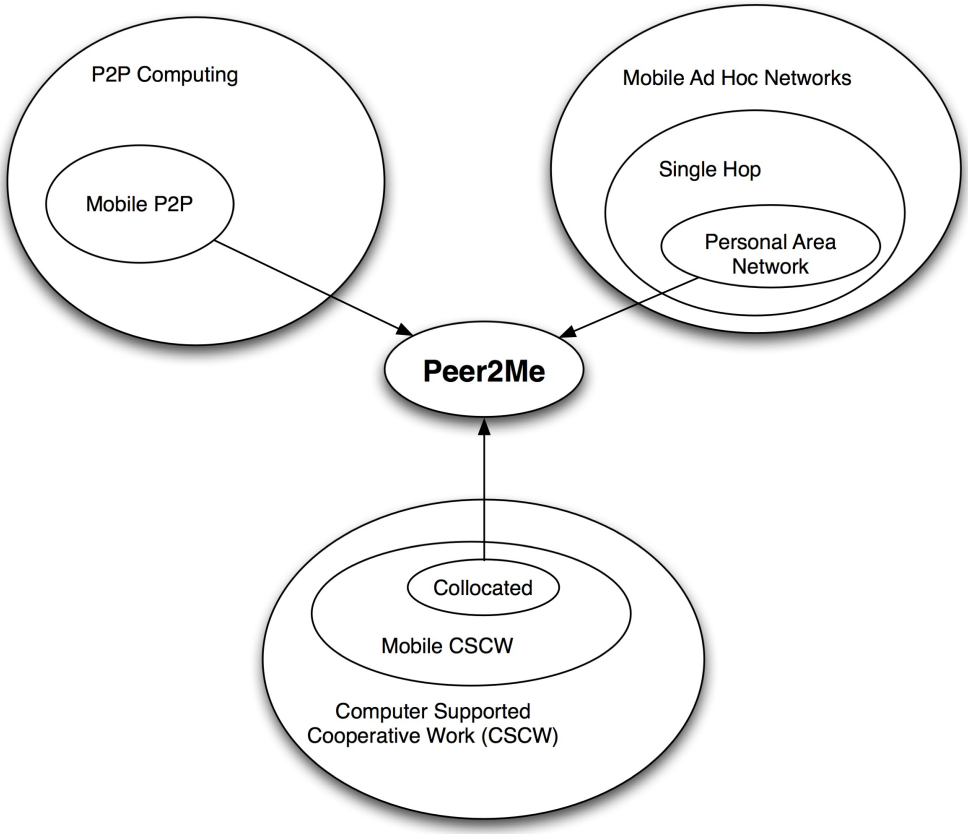


Figure 3.1: The central concepts related to this project.

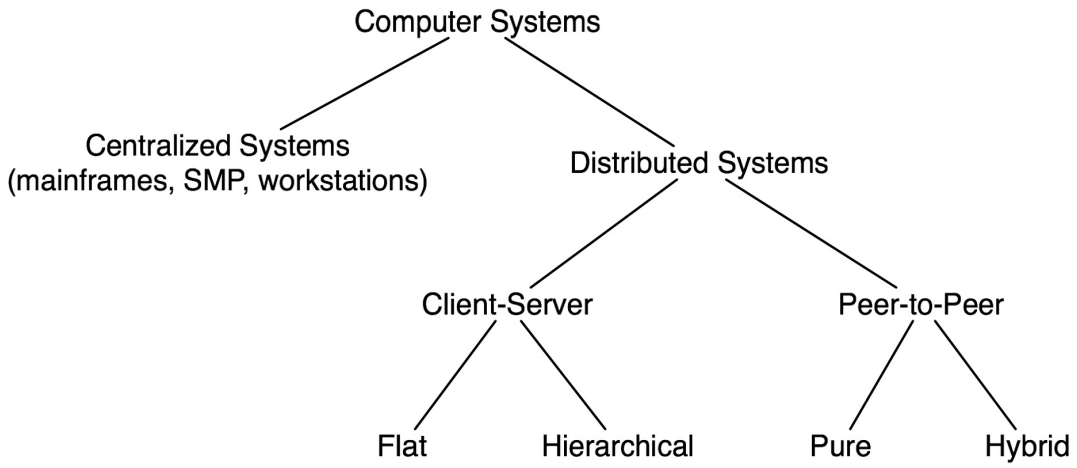


Figure 3.2: Taxonomy of computer systems [57].

Interaction between peers in a P2P network is independent of any central entities. The P2P community have not reached consensus on a definition that covers every aspect of P2P. One way of defining P2P computing is found in [13]:

> "Peer-to-peer computing refers to a class of applications that enables users to form logical networks on top of any infrastructure and to share and exchange digital content."

Mats Thoresen presents a lot of different definitions in [57] and from these definitions he suggests five requirements for a P2P network:

- P2P networks consist of operational computers of server quality. Peers function as both clients and servers.
- P2P networks have an addressing system independent of Domain Name System (DNS). P2P systems need a location independent addressing scheme, i.e. addressing independent of static Internet Protocol (IP) addresses.
- P2P networks are able to cope with variable connectivity, i.e. ad hoc networking. Peers may leave, due to failures or natural causes, and new peers may join the P2P network at any time.
- Peers access other peers and their resources directly, without passing intermediate entities. Once a connection is established between two peers, these two peers communicate directly.
- P2P Networks utilizes the resources at the edge of the Internet. Underutilized computer devices, e.g. home computers, PDAs, cell phones, etc., leverage their collective power.

All these requirements described by Thoresen has influenced the design of Peer2Me. Although Peer2Me is not using Internet as a communication medium, rather some kind of wireless PAN, these requirements lay down the foundation for all types of P2P systems, independent of the underlying transport protocol or medium.

This new way of designing distributed systems offers architectural qualities that differs a lot from the qualities of the classical client-server architectures. We recognize possible advantages of P2P computing as:

Capacity: Utilizing unused resources like bandwidth, storage and processing power on the edge of the network.

Independency: A distributed architecture independent of central entities.

Configuration: Because all peers have the same functionality and responsibility the network becomes more autonomous and self-configurable.

Decentralization: Because functionality and services can be located anywhere in the network, a P2P network does not introduce bottlenecks in the same way as the client-server architecture does.

Extensibility: Adding new resources and making the system grow is not complicated. A peer can join the network and instantly make new services or resources available on the network.

Fault tolerance: There is no single point of failure in a P2P network. The network architecture easily provides a natural replication scheme.

Scalability: An obvious benefit of decentralization is scalability. Consumers of resources also produce resources. If a peer does not need a special service or resource, it could offer it to other peers. In exchange a peer can get needed resources or services from other peers. The scalability of a P2P network is limited by the amount of centralized operations, like synchronization and coordination.

All this advantages will be exploited by applications made with Peer2Me. These applications will be independent of a central server, highly autonomous, fault tolerant, scalable and so on. Peer2Me handles the infrastructure and generalizes P2P functionality to let developers easier make applications that has these advantages.

Although the P2P computing paradigm provides a lot of possible advantages some major problems have to be solved before the full potential of P2P systems can be realized. Three major challenges are described in [16]:

Searching and routing: How to locate resources and services and how to route messages between nodes.

Resource management: Contribution and allocation of resources.

Security and privacy: Prevention from malicious peers and protection of personal information.

There is no central mechanism in a P2P network. This means that peers have to search among other peers to find the resources and services they are looking for. Peers requesting exactly the same resource from the network might communicate with different peers via different routes, with different results. Requests for resources and services might result in an immediate response or might not result in any response at all.

Resource management deals with how peers should contribute and allocate resources. Because there is no central allocation mechanism, the peers have to decide what kind of services and resources they should provide, and how these will be allocated among the peers. The autonomous nature of peers give them the possibility of not to contribute to the network at all, simply taking advantage of other peers sharing their resources. A solution to this problem could be to use concepts from economics, for instance constructing a marketplace, where peers can buy and sell or trade resources and services as necessary. The microeconomic paradigm was used in the design of Mariposa [53], a wide-area distributed database system, here all clients and servers negotiate, buy and sell resources from each other. In this way, queries and responsibility are shared efficiently across the network.

P2P systems, as other computer systems, are exposed to various security threats. Because peers are autonomous, some nodes may be malicious and might threaten the security. Malicious peers might attack the availability of the system, so called denial-of-service (DOS) attack. Methods on

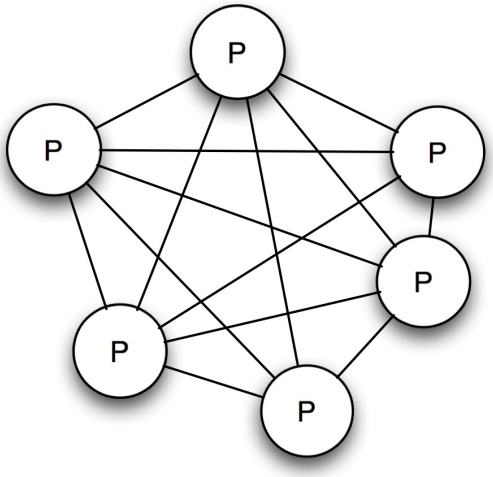


Figure 3.3: Pure P2P Model.

how to trust peers to provide reliable information have to be developed, and users have to be able to control the use of personal information regarding themselves.

There are two core types of P2P network architectures, called hybrid and pure. The key distinction between the two architectures is that the hybrid P2P architecture involves one or more central entity. In a pure P2P network, all peers have the same responsibility. There is no central entity responsible for managing, controlling or coordinating the services and the resources on the network. Because all peers have the same responsibility, any peer in the network can be removed without loosing functionality. Because no central entities exist, much more complex routing and location protocols have to be implemented. However, because of the totally decentralized architecture, applications with high availability, fault-tolerance and good scalability can be built. The Peer2Me framework offers a combination of a pure and a hybrid P2P architecture. Figure 3.3 shows the pure P2P model with peers marked with the letter *P*.

In a hybrid P2P network there are central entities responsible for providing services to peers. These are services like locating resources or routing messages. The central entities are contacted by peers to provide services or exchange information. Figure 3.4 shows the hybrid P2P model with peers marked with the letter *P* and the central entities marked with the letter *S*.

P2P computing provides a completely different way of designing distributed systems. Development using this new paradigm has so far resulted in a broad range of new applications and different architectural approaches compared to old distributed applications. The most famous applications worldwide are file sharing systems like Napster and Gnutella and instant messengers like Microsoft Messenger and ICQ. In [48], Schoder and Fischbach identify five different P2P application categories: Instant messaging, digital content sharing, grid computing, collaboration and web services. Peer2Me is mainly supporting colloborative applications, but may still also cover applications of instant messaging and digital content sharing in ad hoc networks. Since mobile phones together with PAN technology are so different from a stationary computer using Internet for communication, it might be totally different applications that will be adopted and widely

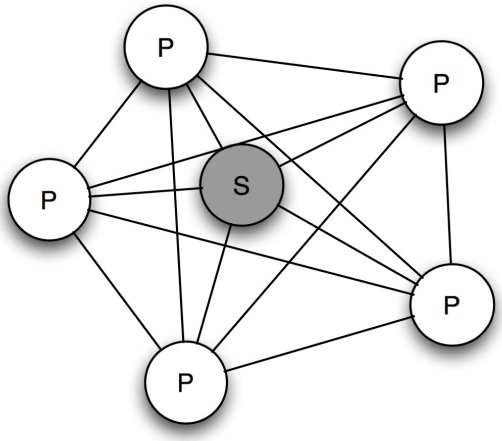


Figure 3.4: Hybrid P2P Model.

used on this technical platform. Mobile phones together with PAN technology are opening totally new possibilities for making new kind of mobile applications covering all different kinds of usage scenarios.

3.1.1 Mobile P2P

Peer2Me is a mobile P2P system. A mobile P2P system should be realized with a pure P2P model. The pure P2P model allow peers to join the network and to discover peer resources without a server infrastructure.

In [35], mobile P2P is defined as follows:

> "A mobile P2P system is a distributed mobile system that consists of mobile hosts that continuously change their physical location and establish peering relationships among each other based on proximity."

A mobile P2P system can be implemented in two different ways:

Without an infrastructure: Using an underlying mobile ad hoc wireless network, like Bluetooth or InfraRed.

With an infrastructure: Using the Internet via a wireless cellular network, like GSM or UMTS.

Peer2Me is designed and will be further evolved to support functionality of a mobile P2P system. The framework will cover applications that do not rely on an existing infrastructure. The framework and its applications will be based upon an underlying mobile ad hoc wireless network. There are clear differences between non-mobile or stationary P2P systems and mobile P2P

systems. Mobile P2P systems are under influences of the general challenges in mobile computing. The main categories of challenges are identified in [22] as: Communication, mobility and portability. Communication challenges are about interruption of transfers, problems with low and varying bandwidth, integration of heterogenous networks and security issues. The challenge of mobility includes difficulties with addressing devices and location-aware information. Portability challenges are about reducing the amount of energy used, the fact that small mobile devices has small screens and user interfaces, low storage capacity and low CPU power. Mobile P2P systems represent challenges that differ from the challenges in non-mobile P2P systems. Some of these challenges are given in [36]:

Resource discovery: The dynamic nature of mobile P2P systems requires more dynamic mechanisms for device and resource discovery.

Data sharing and synchronization: High availability is desirable to provide autonomous peers. To obtain this availability some kind of replication scheme has to be deployed. This introduces the very complex problem of consistency between peers.

All these technical challenges make the design of mobile P2P systems quite difficult. The nature of mobility introduces possibilities along with difficulties. For example, mobile devices are by definition moving, making people able to facilitate ubiquitous computing. Because the devices are moving, they will over time go in and out of range of each other, causing communication links to fail and transfers to be interrupted. The designer has to balance these kinds of contradictions to be able to create useful systems.

3.2 Mobile Ad Hoc Networks

Peer2Me is using some kind of PAN to communicate. All communication between nodes in a Peer2Me application is performed by an underlying PAN. PAN is a subclass of networks called Mobile Ad Hoc Networks (MANET). MANETs are spontaneous, self-configuring, wireless networks with no fixed infrastructure. As devices supporting ad hoc networking are moved around, they are able to detect and connect to other devices that are within a given proximity. When the devices are out of range from each other, the connections are broken. In this way, the devices form spontaneous networks with the possibility of exchanging information. The most widespread definition of a MANET is found in [38] and is as follows:

> "A mobile ad hoc network is a network formed without any central administration which consists of mobile nodes that use a wireless interface to send packet data."

Today's cellular systems, like the Global System for Mobile communications (GSM) and Universal Mobile Telecommunication System (UMTS) networks, rely heavily on the infrastructure. Base stations provide coverage, and services are integrated into the system. This architecture provides good and predictable services, which suits well for cellular telephony. Ad hoc networks have a number of advantages compared to traditional wireless cellular networks listed in [36]:

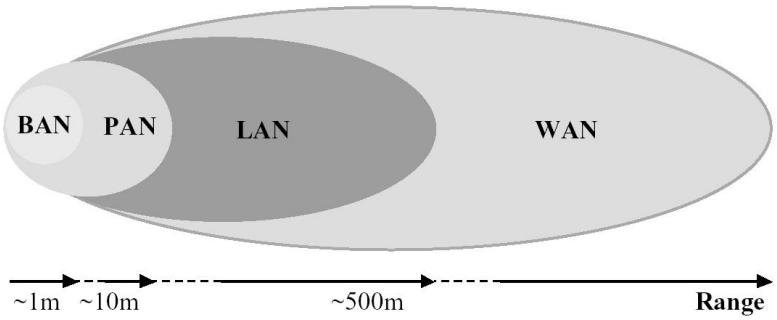


Figure 3.5: Taxonomy of ad hoc networks [15].

No infrastructure required: Ad hoc wireless networks do not rely on wired base stations and for that reason can be deployed in places without existing infrastructure. They can be created spontaneously and on as needed basis, because they require little configuration to setup.

Self-organization: In a wired network the connection topology of nodes is determined by the physical cabling and is therefore fixed. This restriction is not present in an ad hoc network. As soon as two nodes are within proximity of each other, a communication link between them is automatically formed. As a consequence, the network topology of an ad hoc network reflects the relative distance of its nodes and is continuously reconfigured as nodes come within reach of each other.

Fault tolerance: The self-organizing nature of ad hoc networks and the fact that they do not rely on dedicated stations makes ad hoc networks fault tolerant. In a traditional cellular network, a fault in the base, will impair all nodes in its cell. In ad hoc networks, a malfunction in one node can be easily overcome through network reconfiguration.

MANETs can be classified as shown in Figure 3.5. Depending on the coverage range, ad hoc networks are divided into four main classes. As the coverage range increases so does the power consumption, requiring more and more powerful devices. A Body Area Network (BAN) is a network of components distributed on a human body. This could be wearable devices like mobile phones, MP3 players, headsets, microphones etc that are connected with wireless technology. The range of BANs corresponds to the human body range, about 1-2 meters. A Personal Area Network (PAN) connects mobile devices carried by users to other mobile and stationary devices. The communicating range of a PAN is normally up to 10 meters. Wireless Local Area Networks (WLAN) has a range of a building or a part of building, about 100-500 meters. Wireless LANs can be implemented in two different ways. One way is to use some kind of cell-based infrastructure with a centralized controller for each cell. Another way is to use ad hoc networks for devices to communicate directly. A Wide Area Network (WAN) covers a much larger area than the other classes. A WAN may cover areas like a campus or a part of a city.

MANETs can also be classified by the way the nodes form and communicate. We distinguish between singlehop and multihop networks. In singlehop networks (see Figure 3.6), nodes can communicate directly and are within reach of each other. In multihop networks some nodes are out of reach of each other and cannot communicate directly. Therefore, the traffic between these

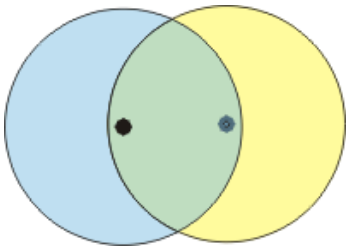


Figure 3.6: A singlehop ad hoc network [1].

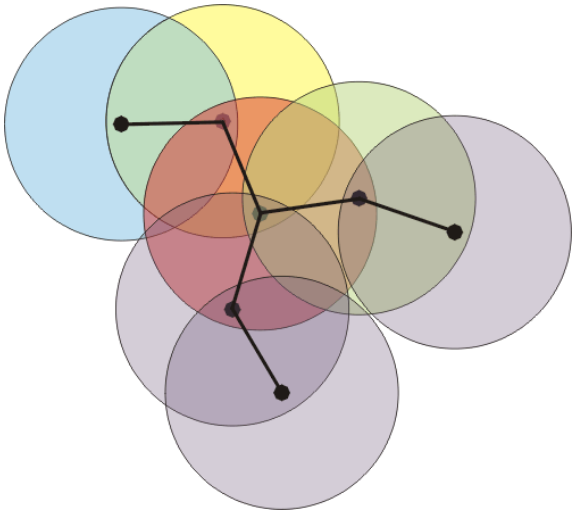


Figure 3.7: A multihop ad hoc network [1].

nodes has to be forwarded by other intermediate nodes. Figure 3.7 shows a multihop network where the communication paths between the far nodes are given by the black lines.

Peer2Me supports collaborative applications communicating using some kind of PAN. A PAN can either be a *piconet* or a *scatternet* (see Figure 3.8). To explain the differences between these to network configurations, we will use the Bluetooth PAN technology as an example. A Bluetooth piconet has a single master device and up to seven slave devices [28]. All these devices are in communication range of each other, therefore all devices need only a single network hop to communicate. The master of the piconet is the one that initiates the connection, and a master in one piconet can be a slave in another piconet. A device in one piconet can communicate with another device in another piconet, making piconets interconnecting into scatternets. Communication between nodes in a scatternet requires advanced multihop communication. For example, a node in a scatternet may have to forward packets on behalf of other nodes. This functionality is not supported in any implemented Bluetooth Application Programming Interface (API). Researchers all over the world are still working on standardizing algorithms and protocols for scatternet communication. Because of this Peer2Me is only focusing on communication between devices within a single piconet.

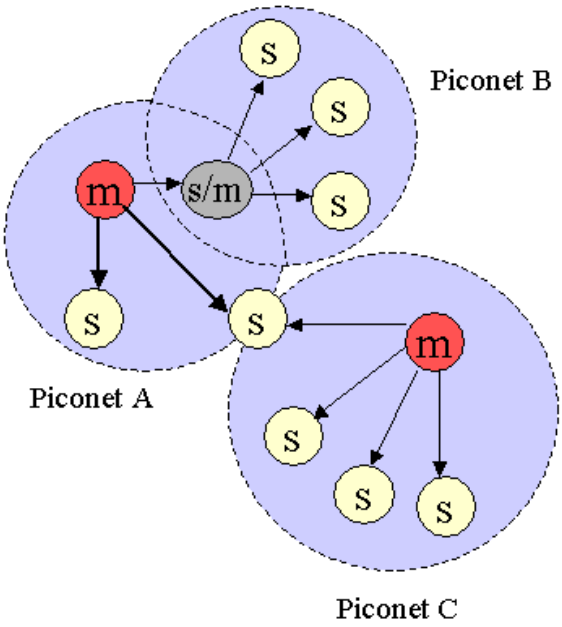


Figure 3.8: A scatternet comprised of three piconets [3].

3.3 Collaboration

Peer2Me is a framework for developing collaborative applications on mobile phones. We will here give a brief overview of the nature of collaboration between humans and different aspects of CSCW to understand the complexity of collaborative applications.

When an activity is to be carried out by more than one individual, issues related to communication and synchronization emerge. This effort can be viewed as a collaborative effort to reach a common goal or share an experience. Human collaboration is a very complex natural system that we do not fully understand today. This vast complexity is due to many contributing factors such as:

Explicit communcation: Person-to-person communication is a thoroughly researched field but still a field we do not completely understand.

Contextual communcation: Much of the information exchanged between the participants in a collaborative activity is of the informal type. This includes gestures, body language, ambiguity in the verbal communication due to the use of irony or sarcasm, etc.

Environment: Often the participants use the environment for communicating, e.g. the use of props when trying to explain a model or example.

This complexity is further expanded by the use of spontaneous and highly unstructured communication paths. As an example consider the setting where to workers meet in the hallway of their office building. They stop and discuss recent advances or problems related to their work. This exchange is more informal than having a well planned status meeting, but can often be more efficient and beneficiary to the participants. Also a less formal setting can give a creativity boost since the participants find stating their opinion less intimidating.

	Real time	Asynchronous time
Same place	I	II
Different place	III	IV

Table 3.1: CSCW dimensions.

3.3.1 Computer Supported Cooperative Work

The CSCW field focuses on the use of computers to support cooperation, and communication in collaborative efforts. During the last decade a lot of research effort has been put into the area of CSCW. In spite of this vast effort, a large number of problems concerning the use of computers for cooperation remain unsolved. In [44], several advantages of collocating a work force is pointed out. Some of these advantages are more efficient communication paths, less ambiguity in communication, more efficient synchronization of work and better knowledge management.

The advantages of being collocated stem from the fact that collaboration is probably the most complex, advanced and unstructured form of human to human interaction as explained above. The technology today is too limited to cope with this complexity and therefore not sufficiently suitable to solve all the problems in the CSCW domain.

According to [21], the domain of CSCW can be comprised by two dimensions, time and place, as shown in Table 3.1. Most of the unsolved problems in the CSCW domain are related to the applications that fall into the "Different Place" category. Using CSCW applications for collaboration between users that are not collocated, makes the application the only communication channel used for collaboration. The users' abilities to communicate are limited by the insufficiencies in the technologies and applications used. In the "Same Place" category, especially coupled with "Real Time", CSCW becomes more of a support for the collaborative effort to enrich or strengthen the processes and communication paths. The Peer2Me framework covers both real time and asynchronous applications in the "Same Place" category.

3.3.2 Mobile Computer Supported Cooperative Work

The mobile collaborative applications developed with Peer2Me supports mobile CSCW. Mobile CSCW can be defined as

> "Working together at variuos sites with the use of mobile IT [60]."

This means that using mobile phones or other portable devices as a platform for deploying CSCW applications places us in the domain of mobile CSCW. Today mobile phones have several advantages when it comes to the area of CSCW.

Mobile phones are highly personal and most users carry their mobile phones with them at all times. This has some major implications on the use of mobile phones for CSCW purposes:

Identification: Since the mobile phones are personal, they can be used to identify a user.

Personalization: A user can store his or her profile on the mobile phone, enabling the mobile phone to function according to the user's specific needs when interacting with other users.

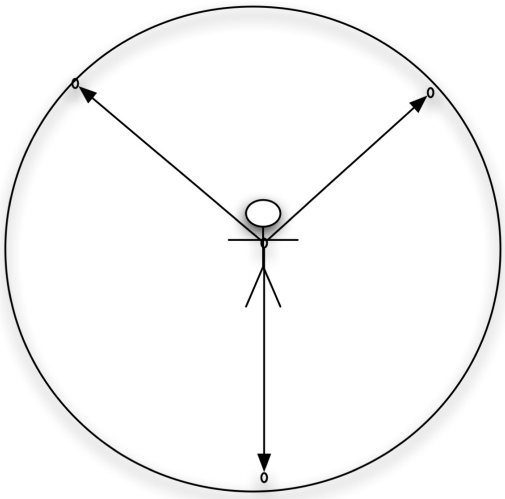


Figure 3.9: A PAN creates a digital sphere around a person.

Availability: Mobile phones can almost be considered to be always on, always present. Due to this, someone using their mobile phone for CSCW purposes will achieve a high degree of availability to other users.

During the last couple of years, mobile phones have started to support more than one transport medium. Still the most important, and the one with the longest range, is the cellular network provided by the telecom operators. During the last years low-range PANs have started to be supported by a number of phones. These ad hoc network technologies enable devices to detect and connect to devices that are in sufficient proximity. This is done in a decentralized manner. These characteristics relate strongly to the nature of human spontaneity, which makes PANs suitable for making spontaneous collaborative applications. A PAN creates a *digital sphere* around a person, see Figure 3.9. This digital sphere is limited by the communication range of the PAN.

When two or more people come in proximity of each other and their mobile devices are within communication range, we define the persons as physically collocated. Their digital sphere will overlap and they will be able to interact (see Figure 3.10). Low-range PANs will force the users to be physically collocated in order to form a limited ad hoc network. P2P communication can be used to enable users to communicate within these networks. P2P networks allow peers to join and leave the network without any configuration, and this fits perfectly with the nature of ad hoc networks. A peer is here defined as the person together with his or her mobile phone. Together, PANs and P2P computing provide the most suitable functionality for building collaborative applications on mobile phones.

By using low-range PANs for mobile CSCW applications, the collaborative efforts will have to be either based on chance encounters between peers (impromptu collaboration) or a planned meeting or gathering of peers (formal collaboration).

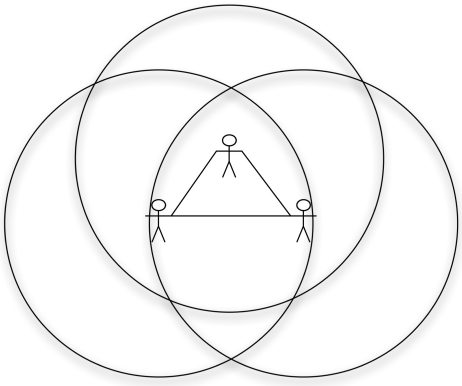


Figure 3.10: Three persons physically collocated.

Impromptu collaboration

Proximity-based ad hoc interaction, made possible by mobile phones and PANs, is referred to in [36] as impromptu collaboration. Impromptu collaboration is recognizable as being:

Opportunistic: The technology enables people to take advantage of opportunities that present themselves.

Spontaneous: The collaborative effort is not planned in any way in advance.

Proximity based: The peers have to be physically collocated.

Transient: The interaction between peers is very short, e.g. a few minutes or seconds.

Impromptu collaboration can involve different degrees of user interaction. To define the broad range of impromptu collaborative applications that can be made with Peer2Me, we define the following three categories of applications:

Requiring user interaction: The application requires user interaction. The users have to explicitly trigger the collaboration activities, start the information search or request a service. Example: Two people at the bus stop that want to exchange MP3 files.

Automatic collaboration: Automatic collaboration between devices. The application is responsible for initiating communication between devices on behalf of the user. The user stores a profile that defines how the application should act with respect to other devices and available services. Example: A person automatically exchanges MP3 recommendations with other people he or she meets when walking around at the campus.

Automatically triggered collaboration: The devices automatically trigger collaboration that requires further user interaction. Example: The mobile devices carried by two different people automatically communicate without user interaction and discover that the two persons are sharing the same taste in music. The two people are alerted and are given the possibility to share MP3 files.

Formal collaboration

Formal collaboration is characterized by being proximity-based, but due to its organized nature it is not opportunistic and spontaneous. This more formal form of using CSCW on mobile phones is more suitable in situations where a collection of users automate parts of their collaborative work process (typically a workflow system).

3.4 Summary

In this chapter we have argued how low-range PANs are suitable for collaborative applications since they force the users to meet face to face in order to interact and how face to face interaction preserves the advantages of collocated work. To further support the spontaneous part of mobile collaboration, the networks formed by the users as they meet should not be dependant upon a central entity for synchronization or network/service provider. Because of this, P2P networks are ideal for the forms of collaboration covered by Peer2Me. Interaction among humans is P2P in nature. In our everyday life, we interact directly with each other, person-to-person, face-to-face, without the need of an intermediate, e.g. when exchanging information or doing some kind of collaborative work. Because of these facts, we argue that CSCW applications should be developed using some kind of P2P architecture. The advantages of CSCW can be combined with those of being collocated by using always on, always present devices. Information is exchanged between users as they physically move close together, or all users are present at a meeting and the devices are used as support to achieve a more efficient work process. These are the basic ideas and concepts that Peer2Me is based upon.

Chapter 4

Previous Work

This master thesis continues the work done in our depth project at the Norwegian University of Science and Technology (NTNU), described in [37]. The depth study was again based on the work conducted by Kirkhus and Sveen described in [32] and [33]. This chapter summarizes the results produced in [37].

4.1 Theoretical Results

Through a thorough study of the Computer Supported Cooperative Work (CSCW) and Peer to Peer (P2P) domains and all types of related research projects, we set our focus to be spontaneous collaboration utilizing ad hoc networks and physical proximity between the participants.

A study and evaluation of available technologies presented mobile phones as the most suitable device for deployment. Java 2 Micro Edition (J2ME) was the most suitable language for implementation due to its platform independence. By evaluating several wireless Personal Area Network (PAN) technologies, we chose to use Bluetooth as a demonstration network medium. Because Bluetooth was supported on more mobile phones than any of its competitors.

4.1.1 Usage Scenarios and Requirements Engineering

In [37], we presented some usage scenarios for mobile collaborative applications. These scenarios were classified using a classification matrix proposed by us. This classification matrix is shown in Table 4.1.

The vertical dimension separates applications using routing algorithms for sending messages via other nodes and applications only sending single hop messages.

Multi-hop: The scenario is based on functionality that enables a search for required resources conducted on a large network of nodes and subnets tied together in large scatternets. Users can, network-wise, be located with long distances between them and messages may require multiple hops to reach their targets.

User Interaction/ Advanced P2P functionality	User	Auto	Hybrid
Multi-hop			
Single-hop			

Table 4.1: Classification Matrix, [37]

Single-hop: The scenarios is based on the exchange of information or the use of services on devices located in a local, temporary ad hoc network. The users are, network-wise, close to each other and messages are transmitted using single hops inside a piconet.

The horizontal dimension classifies the applications by looking at to what degree they require user interaction.

User: The scenario requires user interaction. The user has to explicitly trigger the collaborative activities, start the information exchange or request a service.

Auto: Automatic collaboration between devices. The application is responsible for initiating communication between devices on behalf of the user. The user stores a profile that defines how the application should act with respect to other devices and available services.

Hybrid: A combination of User and Auto where devices automatically trigger collaboration that requires further user interaction.

We described and analyzed the following five scenarios to uncover requirements for a framework supporting collaborative applications on mobile phones. The five scenarios were:

1 - Converging Top Ten List This scenario describes how users in a population can register information in a top ten list form on their mobile devices. When they meet with other users from the same population, their lists are compared causing each user's list to converge towards the true top ten list. The example used is students getting a list of the best beer prices in the campus area.

2 - Tourist Information This scenario describe how people in a city can collect, provide and exchange digital information about their city that tourists may be interested in. When new tourists come to town they can obtain useful information about sights, restaurants, useful telephone numbers, etc. All this happens automatically, when tourists pass other people on the streets, the information the tourists are interested in is downloaded from the devices owned by the passing people.

3 - Ubiqutous Flea-Market If a person wants to buy or sell something, instead of placing an ad in the local newspaper he can register the item of interest on his mobile phone. When he goes to the grocery store, the movies, etc his phone will automatically connect to other buyer/seller phones and check if an interest match can be made. If a match is made, then the seller's contact information is downloaded to the buyer's phone and the buyer can call the seller when he gets home.

User Interaction/ Advanced P2P functionality	User	Auto	Hybrid
Multi-hop			
Single-hop	5	1, 2, 3	4

Table 4.2: Classification Matrix with scenarios., [37]

4 - Planning the Next Meeting When several people from different companies end a meeting by setting a date for the next meeting, finding a date that suits everyone can be a tedious effort. If all the participants have their calendars on their mobile phones or another portable device, then the meeting chair can set his phone to gather everyone's free time and calculate the most suitable date.

5 - PAN Instant Messaging Instant messaging has grown to become almost a ´´killer-application" on different mediums and is also one of the most common P2P applications. In this scenario two students are attending a lecture at their university. Both students are very familiar with the current topic, but need to stay because of some information that will be given later in the lecture. To pass the time they establish a chat channel with their mobile phones and spend the first part of the lecture talking with instant messages.

These scenarios were then classified using the classification matrix, shown in Table 4.2. The table shows no scenarios in the Multi-hop dimension. In [37] we defined multi-hop communication to be beyond the scope of the depth project and did not focus on such scenarios. This scope limitations still applies.

The most relevant scenarios were then analyzed to uncover the requirements for a framework supporting the development of mobile collaborative applications using ad hoc networks. These requirements will be presented in Chapter 7.

4.2 Technical Results

From our gathered requirements and our design, we started an implementation of the framework and a Bluetooth network module. The framework and the network module was tested by developing simple test applications utilizing as much of the framework functionality as possible.

The final code did not cover all of the stated requirements and had some major flaws. One quite large problem we had was that an application where the interaction between to devices required a user to press a button would only run on the Nokia 6230 Emulator software and not run on any of the mobile phones we had available for testing. When using the mobile phones, the application would freeze and the phone needed to be reset. Another problem we didn't discover until after we had finished the project ([37]) was that the implementation we had was unable to connect more than two phones. In order to discover this problem we needed a third phone which we did not get until January 2005.

4.3 Remaining Work

As stated in the end of [37], some work remained on the framework. The main issues we wanted to address were:

Functionality: Not all of the functional requirements were fully implemented when we finished the Peer2Me prototype. One rather large part of our further work would be to implement more of the requirements. Two major issues addressed were:

- Support for more message formats. Currently only text messages are supported.
- Creating support for message routing between the slaves in a piconet.

Optimalization: A lot of the calls in the prototype are blocking calls. The number of blocking calls should be kept to an absolute minimum and the degree of parallelity should be as high as possible.

Reliability: The prototype was unstable, so one quite important issue was to improve it with respect to stability and fault tolerance.

Building applications: In order to fully test the Peer2Me framework, scenario applications should be developed and tested to check the suitability of the framework and the chosen technologies.

Chapter 5

State of the Art

In [37], we did a thorough review and evaluation of all projects related to or similar to the Peer2Me project. We will not repeat all of these results here, but evaluate any new projects we have come across and give a quick summary of the projects evaluated in [37].

5.1 BEDD

BEDD is a software suite, developed and maintained by the BEDD Corporation, [17]. Originally BEDD was created as a dedicated device running only BEDD software, but was later migrated to the Symbian Series 60 OS to allow it to be installed and run on smartphones. Unlike the other related projects we have evaluated, BEDD is a commercial project and not a research or open source project. The BEDD software was first released in Singapore in May 2004. Future releases are planned in Asia, Europe, Middle East, Africa and America. Although BEDD currently is implemented as a native application, the BEDD Corporation is working on a Java 2 Micro Edition (J2ME) based client.

BEDD uses Bluetooth as a transport medium for exchanging messages between devices that are running the BEDD software and that are within Bluetooth range of each other. Since BEDD is a commercial software suite, there is not much information available as to how the software actually works, but judging by the number of different modes BEDD can run in, we would guess that BEDD has some sort of core that handles the basic ad hoc network implementation with the specific applications using this core to communicate.

Below, we list and describe some of the applications included in the BEDD software suite:

BEDDmates: BEDDmates is an application that connects to other devices running BEDD as they move within reach and exchange user profiles. All profiles that are downloaded to a device is stored on the device and made available for browsing later.

The profiles that are downloaded are automatically matched with the local profile and the user can browse to see if any of the downloaded profiles match his criterias in his search for friends and or life partners. If the user decides to keep all downloaded profiles stored on his or her device, the downloaded profiles can also be used to match against other devices the

user encounters. This way, profiles can travel through the network and cause matches far away from their home device.

BEDDbay BEDDbay is an application that allow users to register ads for buying, selling, renting or trading items. When devices running BEDD come within reach of each other, all ads available on the devices are transferred.

BEDDtalk BEDDtalk is a live chat application for devices running BEDD. It supports both group conversations such as chat rooms, personal chat channels and broadcasting of messages.

5.1.1 Evaluation

Of all projects and products we have found that somehow relate to Peer2Me, BEDD is probably one of the closest ones. The three applications we have described above are very similar to three of our own scenarios, presented in Chapter 4.1.1.

The downside of BEDD is that it is a commercial product and not available for anyone to use, neither as a developer framework or as end user applications. However, BEDD proves that there is in fact a commercial market for services such as the ones described in our scenarios.

5.2 JSR-259: Ad Hoc Networking API

The JSR-259 Ad Hoc Networking API is a Java Community Process (JCP) initiated by mobile phone vendors such as Siemens, Nokia and Panasonic, [4]. The aim of the JSR-259 is to create a standard Application Programming Interface (API) for communication between nodes in an ad hoc network, implemented in J2ME. The API will allow third party vendors to develop P2P applications for mobile phones.

According to [4], the finished API will feature methods for:

- Service Discovery
- Service Registration
- Service Availability Alert
- Service and Service Capability Inquiry
- Remote Service Consumption

5.2.1 Evaluation

The JSR-259 is a going to be a class library available for application developers, just like Peer2Me. However, Peer2Me has a head start on the JSR-259. The JSR-259 was initiated in November 2004 and the final draft of the specification release date is set to late in the 4th quarter of 2005.

The Java Community Process is a quite tedious process and takes a long time to finish, in addition to this it takes a long while before a new specification is implemented and included on commercially available devices.

The goals of the JSR-259 overlaps with the goals of the Peer2Me framework, but the JSR-259 will not be supported by a large enough number of devices to be useful for at least 1.5 - 2 years. Peer2Me's goal is to be useful for third party developers within the next 6 months.

5.3 Rocky Road

RockyRoad is a P2P protocol for both J2ME and J2SE [5]. According to the RockyRoad website it is also the only P2P protocol that works on both platforms. The RockyRoad implementation is directed at wireless networking technologies and currently have support for TCP or UDP over IP, SMS/USSD and GPRS over GSM or TDMA.

The RockyRoad implementation is based around a central entity called the core. The core entity offers a packet oriented P2P protocol for pure P2P systems. In addition to the pure P2P protocol, RockyRoad also has support for the concept of *privileged peers*. A privileged peer is a peer that can hold information related to certain services such as location of peers, indexing services, security, etc.

According to the RockyRoad white paper, [2], RockyRoad is scalable, and new transports or packets can be added without changing the RockyRoad core.

5.3.1 Evaluation

There are many aspects related to RockyRoad that are very interesting when seen in comparison to the Peer2Me framework. The notion of modular transports is also used in Peer2Me. Such modularity makes a developer framework more useful. If the framework does almost what you need, but uses the wrong network transport medium, a new transport module can be created and then enable the framework for new uses.

One of the main differences between RockyRoad and Peer2Me is the focus on protocol versus complete network infrastructure. RockyRoad's core entity enables nodes to communicate via a specified protocol while Peer2Me offers, in addition to protocols for sending messages, group management and built in support for service discovery.

5.4 Other Projects

In this section we will give brief summaries of the projects we evaluated for the writing of our depth project last year [37]. For a more detailed evaluation of these projects refer to [37].

iClouds As described in [26, 25], iClouds is an architecture for impromptu collaboration and spontaneous exchange of information between users that come within each others digital

sphere. The goal of iClouds is to provide a platform for spontaneous collaboration, taking advantage of the observation that people gathered at the same location often have common interests or goals.

IPAD/Hummingbird IPAD is a device designed for interpersonal awareness and collaboration, described in [27]. IPAD/Hummingbird is not a generic platform but a specialized prototype and can not be used for other scenarios than those already built into it.

JXTA JXTA is a Java framework for creating P2P networks. Two different projects exits for porting JXTA to wireless mobile devices, JXME proxied and JXME proxyless. JXME proxied relies on a central device working as a proxy between nodes in the network and does not cover all the needs of a mature mobile P2P network. JXME proxyless is an attempt to create a fully matured version of JXTA on mobile phones. No implementations of JXME proxyless is available with support for PAN, only TCP/IP.

PROEM Proem is an open computing platform that provides a complete solution for developing and deploying P2P applications for MANETs, [34]. The platform is based on experiences from developing a series of mobile applications. PROEM is implemented as a platform independent framework using J2SE. Todays mobile devices only support J2ME and can therefore not use the PROEM platform for developing P2P applications.

The Spectre Framework The Spectre framework, [33], was designed as a Master Thesis at IDI, NTNU by Sveen and Kirkhus during the spring of 2004. No implementations of the Spectre framework exist and will probably never be available. However, the design of the Peer2Me prototype described in [37] is inspired by the Spectre design and draws on a lot of the rationalities behind the design decisions made in [33].

5.5 Conclusions

After a thorough mapping and evaluation of projects similar to or overlapping with the Peer2Me project, it seems that none of them can be used for answering our research questions. The closest we get to a reusable project is the BEDD software suite, but this is a proprietary project with closed source and commercial interests. In addition the JSR-259 can also be used for the same purposes as the Peer2Me framework, however the JSR-259 will not be available on any devices for at least another year, probably more.

The limitations inherit in the available related projects have led us to the conclusion that we need to develop our own framework in order to answer the questions posed in Chapter 2.

Chapter 6

Technology

This chapter reviews and updates the technology study performed in [37]. We start by discussing mobile phones as deployment platforms and then move on to review the supporting technologies that Peer2Me rely on.

6.1 Mobile Phones

In [37], we found that mobile phones were the most suitable device to deploy Peer2Me on due to their spread in the population. No new device classes that can be used for deploying mobile collaborative applications have been released since then. This, and the fact that mobile phones keep evolving and new functionality such as MP3-players, larger storage capacity, etc are built into the mobile phones convinces us even more strongly that they indeed are the right devices for deployment.

6.2 Java 2 Micro Edition

In [37] we decided on Java 2 Micro Edition (J2ME) as the most suitable implementation platform. Since then, no other platforms have been introduced, but there is currently a work in progress that will update parts of J2ME. We will now give an updated overview of the J2ME platform.

6.2.1 J2ME Architecture

J2ME is basically Java for mobile devices, which means that it is the Java language stripped down to run on devices that are inferior to modern computers when it comes to CPU frequency, memory size and storage capacity. The architecture of J2ME is built with a layered approach shown in Figure 6.1, as described in [39]. The bottom part of J2ME is its virtual machine (VM) called the Kilobyte Virtual Machine (KVM). All applications running on J2ME are executed by the KVM. The basic Application Programming Interfaces (APIs) available when running programs through the KVM are the classes defined in the Connected Limited Device Conifuration (CLDC). In Figure

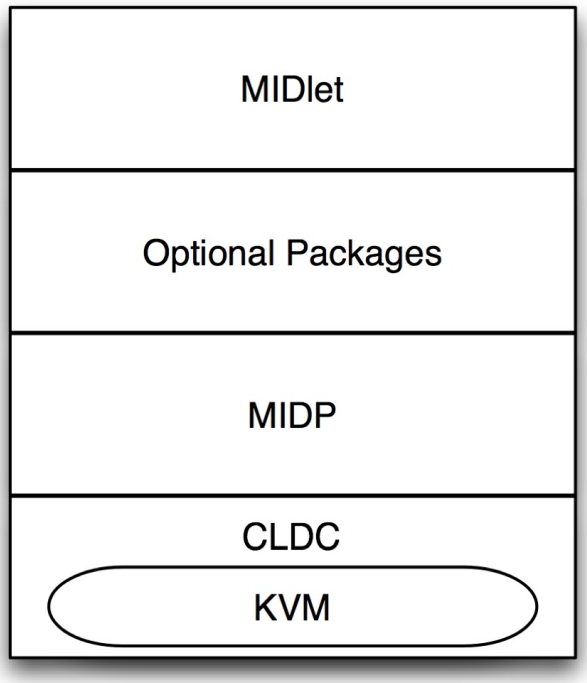


Figure 6.1: Architectrual overview of J2ME.

6.1 the KVM is placed inside the CLDC box, this is due to the two components' tight coupling. CLDC defines generic classes for network connections and utilities that are common across a quite large family of devices. The next layer of the architecture is the Mobile Information Device Profile (MIDP) which defines a common API for the mobile device family, e.g. mobile phones.

In addition to the basic components of J2ME described above, optional and/or third party libraries can be included as well. The Bluetooth APIs, XML parsing libraries, etc are examples of such packages. The Peer2Me framework will also become one such library that can be included as a third party library for applications.

The top level of the J2ME architecture are the MIDlets. A MIDlet is an application written for J2ME using the MIDP class library. All MIDlets must be subclasses of the MIDlet class defined by the MIDP specification.

6.2.2 Future Releases

At the moment CLDC exists in version 1.0 and 1.1, but no mobile phones supporting the 1.1 version have been released to the consumer market yet. We will probably see mobile phones supporting the 1.1 version in the near future.

MIDP is currently available in version 1.0 and 2.0, although 1.0 is discontinued from Sun. Almost all Java enabled phones released this year support MIDP 2.0. The next generation of MIDP is

already on its way. On March the 22nd 2005, The Java Community Process (JCP) released a new Java Specification Request (JSR) aiming to define MIDP version 3, [46].

The MIDP 3.0 overview is quite promising and will take Java on Mobile phones to a new level giving support for, among other things:

- Enable multiple concurrent MIDlets in one VM.
- Specify proper firewalling, runtime behaviors, and lifecycle management issues for MIDlets.
- Enable MIDlets running in the background.
- Enable "auto-launched" MIDlets (e.g. started at platform boot time).
- Enable shared libraries for MIDlets.

The features listed above enable developers to create MIDlets that run as services in the background. This is a major improvement compared to how J2ME works today. For instance, if the Peer2Me applications that handle automatic exchange of information is to have any real value for the end user they need to be as unintrusive as possible. To accomplish this, the application should be put to run in the the background constantly. Setting MIDlets to execute in the background is not possible with MIDP 2.0, so the introduction of this ability will be a big advantage for the Peer2Me framework.

6.3 Wireless Personal Area Network Technologies

Peer2Me is framework for developing mobile collaborative applications utilizing some kind of Personal Area Network (PAN) to communicate. The layered architecture and the module-based design makes the framework independent of the underlying network technology. This means that developers can make applications that are portable and that can be used on different underlying network technologies. This change in underlying network technology can even be done during run-time. Applications made with Peer2Me becomes abstract and independent. Peer2Me currently only supports one network technology, Bluetooth, but the popularity and spread of network technologies on mobile phones are rapidly changing. In [37], we did a thorough evaluation of all existing PAN technologies to be able to choose one demonstration network medium. In this section we will describe what has happened in the area of wireless PAN technology since our last project. This is because we want developers to have an updated report on the available PANs that could be possible to implement as a network module in Peer2Me. The section will for each PAN give an update on specification, the spread of the technology on mobile phones and available APIs and developer tools.

6.3.1 Bluetooth

Bluetooth is a wireless PAN with a communication range of about 10-100 meters and a communication speed of about 721 Kbps. In this section we will present the news related to the Bluetooth technology.

Specification

The current version, version 1.0, was ratified in March 2002. Both the Bluetooth EDR specification and the Bluetooth 2.0 specification is emerging, read more about these new standards in [37].

Support on Mobile Phones

According to [9] mobile phone producers estimate that about 30-40 percent of all existing mobile phones supports Bluetooth technology. Soon up to 50 percent of phones shipped will include Bluetooth, and Bluetooth products targeted at the mobile phone market has a larger market share than all other product types together. By 2007, it is estimated that over 70 percent of all mobile phones supports Bluetooth technology. These estimates proves the market position and wide spread usage of Bluetooth technology on mobile phones. To be able to run Peer2Me applications on a mobile phone it is not only necessary to support Bluetooth technology itself in hardware, the software on the phone must implement the Java APIs for Bluetooth Wireless Technology (JABWT) as well. A list of which phones that support this API is given in the next section.

Programming APIs and Tools

There are many implementations of the Bluetooth specification (or Bluetooth Stacks) available. Each of these implementations is necessarily based upon a proprietary API, usually a API written in C. This makes developing Bluetooth applications using these APIs a very complex task. JABWT provides a solution to these problems. JABWT provides a standard API for programming Bluetooth applications. This standard is developed by the JSR-82 Expert Group. JABWT is an optional package in the second layer from the top in the J2ME architecture (see Figure 6.1). The JABWT provide key abstractions that simplify the tasks involved in building a Bluetooth application.

Table 6.1 shows which mobile phones that include the JABWT, Bluetooth API, and thereby can run Peer2Me with the included Bluetooth network module.

6.3.2 ZigBee

ZigBee is a wireless PAN with a communication range of about 30 meters and a communication speed of about 250 Kbps. It was originally designed as a wireless process and control network, but is also targeted at consumers as a PAN. In this section we will present the news related to the ZigBee technology.

Specification

According to their website[1], the ZigBee Alliance ratified the first ZigBee specification in the middle of December 2004. The specification is a result of two years of work contributed by about

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

Manufacturer	Model	Available for purchase
Nokia	6600	Yes
Siemens	S65	Yes
Siemens	S66	Only in the US
Siemens	S6C	Yes
Siemens	S6V	Yes
Siemens	SK65	Yes
Siemens	SP65	Not yet
Sony Ericsson	P900	Yes
Sony Ericsson	P908	Yes
Sony Ericsson	P910a	Not yet
Sony Ericsson	P910c	Not yet
Sony Ericsson	P910i	Not yet

Table 6.1: Mobile phones with Bluetooth API.

100 worldwide companies. The ratification of the specification enables manufactures of ZigBee technology to verify and obtain ZigBee-compliant certification and later release their products on the market.

Support on Mobile Phones

The mobile manufacture Pantech and Curitel showed at the ZigBee Alliance Festival at the 8th of December 2004 a prototype that featured ZigBee technology. Pantech and Cruitel are using the ZigBee technology to make applications for home networking. The phone includes functionality for controlling electrical appliances, monitoring temperatures,etc.

Programming APIs and Tools

There are no developer tools or programming APIs for Java or mobile phones.

6.3.3 Wireless Firewire

Since the 1394 Trade Association[2] approved a proposal to create a wireless protocol adaption layer (PAL) to use the 1394 protocol (Firewire) over the IEEE 802.15.3 wireless standard [10] [30] in May 2004, there has not been any news related to wireless Firewire.

6.3.4 Wireless USB

Wireless Universal Serial Bus (USB) is a new PAN technology currently under development. It is planned to have a range of 2-10 meters and a communication speed of 480 Mbps. In this section we will present the news related to the wireless USB technology.

[2] http://www.1394ta.org

Specification

The IEEE organization has the responsibility to develop the Ultra Wide Band (UWB) standard that Wireless USB is supposed to be based upon. The industry is still waiting for IEEE to release the standard.

Support on Mobile Phones

According to [6], the electronic manufacture Freescale Semiconductor in partnership with Motorola have developed a prototype of an UWB-enabled mobile phone. This phone was demonstrated at the Consumer Electronics Show (CES) in Las Vegas, USA, on the 4th of January 2005. At the same event it was made clear by several participants of the official UWB Forum[3] that mobile phones supporting UWB will not be a reality in the consumer market until at least 2006.

Programming APIs and Tools

The Java Community Process has defined a Java Application Programming Interface (API) for Universal Serial Bus (USB) communication. The target platform is both the Java 2 Standard Edition (J2SE) and Java 2 Micro Edition. This Java Specification Request is called JSR 80: Java USB API. The specification provides a Java API for communicating with devices attached with USB. It allows Java applications to discover, read, write, and manage USB devices. There is not yet any support for this API on any mobile phones. However this API can be included as a J2ME optional package if any mobile phone vendors decide to produce phones with WUSB.

6.3.5 Wireless Local Area Network

Wireless Local Area Network (WLAN) is a technology for creating wireless networks using the same protocols as the Internet. In this section we will present the news related to the WLAN technology.

Specification

There is no news or known changes related to the 802.11 standard, that the WLAN technology is based on, since our last project.

Support on Mobile Phones

Japan's biggest mobile phone carrier, NTT DoCoMo, released in November 2004 a phone with WLAN support for the Japanese market, called N900iL and made by NEC. There is although no sign of a release of this phone in the european market.

[3] http://www.uwbforum.org/

Royal Philips Electronics announced March 2005 a new low-power 802.11g wireless LAN semiconductor solution for the mobile phone market [8]. Philips claims that this new solution enables communication through WLAN networks up to five times faster than current 802.11b products on the market without compromising the battery life of the mobile phone. Since 802.11g and Bluetooth use the same frequency spectrum, Philips has made an integrated hardware interface and algorithm that allows the WLAN and Bluetooth technologies to share spectrum without sacrificing quality or performance.

Programming APIs and Tools

Ad hoc WLAN programming functionality is available for the Symbian Operating System. Symbian is the most popular and widely used operating system for mobile devices. The APIs for ad hoc WLAN programming is a quite new technology and is described in a technical report from Nokia released in April 2005 [42]. Unfortunately, there are no ad hoc WLAN support in J2ME.

6.3.6 Bluetooth and UWB Cooperation

On the 5th of May 2005 the Bluetooth Special Interest Group (SIG) announced their intent to cooperate with the developers of the Ultra Wide Band (UWB) to combine the strengths of both technologies. The strength of UWB is that it has a very high transfer rate, as high as 480 MBps, which makes it the fastest available wireless technology for PANs. This will open for applications that require transfer of large amounts of data, for instance high quality video. These properties will be combined with the strengths of the Bluetooth technology. The main strengths of Bluetooth is the maturity of the technology, its spread and large user base, the qualification program and a comprehensive and widely used application layer.

A combination of UWB and Bluetooth could definitively evolve to the number one wireless PAN technology and may become the true standard for PAN communication. The main problem of this project is that the cooperation between two organizations of this size takes a lot of time and requires a lot of conflicts and disagreements to be resolved. In a press statement, the leader of the Bluetooth SIG, Michael Foley, estimates the first resulting products of the cooperation to hit the market “in a couple of years”.

Part II

The Peer2Me Framework

Chapter 7

Requirements

This chapter lists the requirements we have found for the Peer2Me framework both through our work in this project as well as our work in our previous project described in [37]. The new requirements added during this project are a result of the evolution of the framework throughout our explorative and iterative development life cycle, described in Chapter 2.2.1.

7.1 Functional Requirements

In this section we will describe all the functional requirements that we found for the Peer2Me framework.

7.1.1 Previously Gathered Requirements

Through our depth-project, described in [37], we gathered a set of requirements for the Peer2ME project. This report documents the continuation of our depth-project and is based on the same requirements.

7.1.2 Complete List of Requirements

If we combine the requirements from [37] with the requirements introduced during our work on this report, we get the complete list shown in Table 7.1. New requirements are marked with a "Y" in the *New* column. A more thorough description of the new requirements are given in the next section. The requirements that were implemented and tested in the previous project are marked with a "Y" in the *Covered by prototype* column. Requirements that where partially fulfilled in the previous project are marked with a "P" in the same column.

7.1.3 New Requirements

When analyzing the scenarios with respect to the actual applications, we found that almost all of them required some sort of persistent data layer. Although Peer2Me is a framework for supporting

Requirement:	Requirement:	New:	Covered by prototype
1	FR1: The system must support mobile phones.		Y
2	FR2: The system must support creation of ad hoc networks.		Y
3	FR3: The system must be able to connect to an existing ad hoc network.		Y
4	FR4: Nodes in a network must be able to exchange messages.		P
5	FR5: The system must be able to create a group of nodes related to a specific application.		Y
6	FR6: The system must support multicasting and broadcasting of messages within a group.		N
7	FR7: The system must be able to search for other phones supporting the same service.		Y
8	FR8: The system must support the creation of groups as closed or open.		N
9	FR9: The system must support to allow a node to try to join a closed group.		N
10	FR10: The system must allow users in a closed group to reject other nodes to join the group.		N
11	FR11: The system must support to allow a node to join an open group.		P
12	FR12: The system must be able to present decision messages to the user.		N
13	FR13: The system must be able to present information to the user about framework related events.		N
14	FR14: The system must be able to support different kinds of network mediums.		Y
15	FR15: The applications must be independent of what network medium that is currently in use within the system. The application that is using the system to handle network traffic should not have to know what kind of network medium that is used by the device or make any kinds of adjustment to fit a specific network implementation.		Y
16	FR16: The system must be able to identify where a transfer originated from. To be able to send direct replies to a given device, it must be possible to see where a transfer originated from.		Y
17	FR17: The framework must include a mechanism for storing objects.	Y	N
18	FR18: The framework must include a mechanism for retrieving stored objects.	Y	N

Table 7.1: Functional requirements.

the use of ad hoc networks, we decided to include requirements that deal with persistence handling in the requirement set. The persistence requirements are:

FR17: The framework must include a mechanism for storing objects.

FR18: The framework must include a mechanism for retrieving stored objects.

7.2 Non-functional Requirements

In [37], we described two non-functional requirements for the Peer2Me framework prototype. One of our goals in this project is to increase the overall quality of the Peer2Me. Wireless networks are more unstable than wired networks. For instance, interference between networks can occur and physical obstacles may prevent data transfer to complete. In order to achieve a better quality of services provided by the framework, we will add the following non-functional requirements:

NFR3: The framework must adapt to errors that arise due to the unstable nature of wireless networks.

Also, during the writing of [37], we did not have a complete grasp of the concept of master and slave in general PAN network topology. The concept of master and slaves in a network forces all messages between slaves to be routed through the master node. Some applications may send information not intended for all network members, and the framework should therefore make sure that nodes do not have access to messages not intended for themselves. This gives us a new non-functional requirement:

NFR4: The framework must prevent applications from getting access to messages not addressed to them.

Table 7.2 shows the complete list of non-functional requirements for the Peer2Me framework. Requirements that are new in this project are marked with the letter “Y” in the *New* column.

Requirement:	Requirement:	New:	Covered by prototype:
1	NFR1: The framework must be able to transfer messages fast enough for real time interaction. By fast enough, we mean that normal length text messages should give the impression of appearing instantly on the remote phones.		N
2	NFR2: The framework must be able to detect the disconnection of nodes within a group and notify relevant applications and nodes about this		N
3	NFR3: The framework must adapt to errors that arise due to the unstable nature of wireless networks.	Y	N
4	NFR4: The framework must prevent applications from getting access to messages not addressed to them.	Y	N

Table 7.2: Non-functional requirements.

Chapter 8

Design

This chapter describes the design of the Peer2Me framework. First, we give a quick overview of the central concepts of Peer2Me, then an architectural overview and a detailed design where changes from the original prototype design described in [37] are mentioned specifically.

8.1 Domain Concepts

In our design, there are a few concepts that are essential for understanding the system. We will now provide a description of these concepts and the relationship between them.

First we will describe each concept by itself. We will then move on to explain which parts of the domain model that will need a network specific implementation, and how an application using the framework can utilize the components.

Framework: The framework entity is the core entity of our framework and will be used as an interface between the application and the rest of system. It manages resources such as known peers, available network mediums, etc.

Node: A node is a logical representation of a peer, in our case a mobile phone running the framework.

Network: The network is an abstraction of the network layer. The applications will usually not use the the network instance directly but access it through the framework instance.

Service: A service is connected to a specific application and supported by zero or more nodes.

Group: A group is a collection of nodes providing the same service and communicating using a homogeneous network. Every group is required to have a master node administering the group.

Message: A message is the entity that can be exchanged between nodes connected in a group. A message can either be sent to a specific node or to a group as a whole.

Application: An application is the software using the framework to present a service to a user.

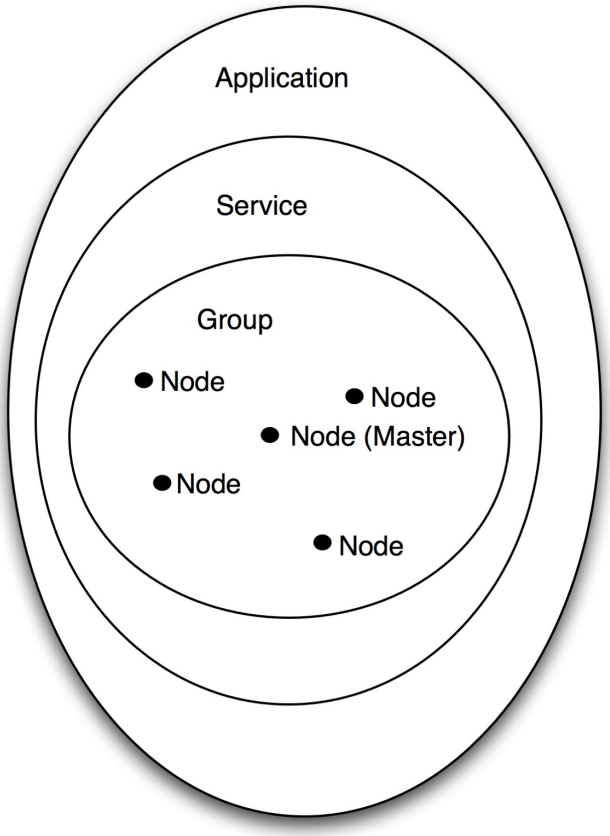


Figure 8.1: Peer2Me domain concepts.

Of the seven concepts described above, two are dependent of network technology: The network and node entities. We will create abstractions of these that are independent of the network technology, thus hiding the network specific issues from the application and the framework itself. The group entity will in some cases require a network specific implementation, depending on the capabilities of the underlying network.

In Figure 8.1, the relation between the domain concepts Application, Service, Group and Node is shown. An application provides one Service to its peers. Along with this service there is an associated Group. A group consists of several Nodes with different roles. One Node will always have the role as Master, this is the administrator of the group. Other nodes, working as slaves, will have to route messages in the network via the master device.

8.2 High Level Architecture

When designing a framework, one of the key challenges is to create a suitable high-level architecture. The main goal of our architecture is to make the framework as independent of network technology as possible. This will reduce the work needed when migrating the framework to a new

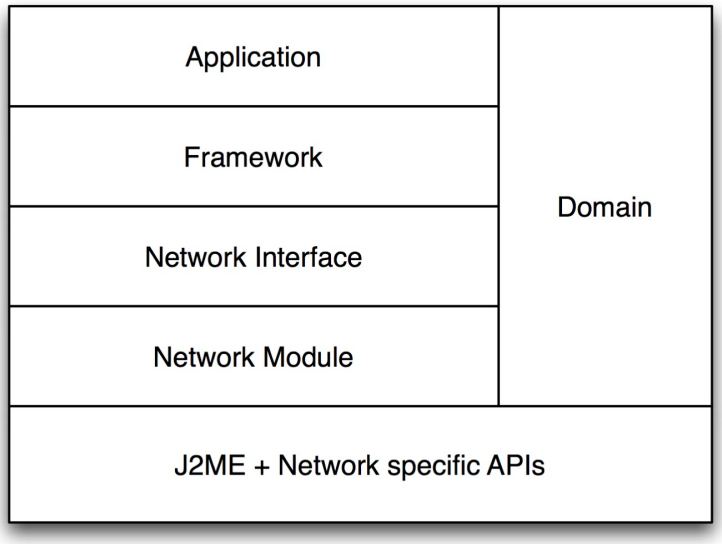


Figure 8.2: Architectural overview of the Peer2Me framework.

network medium. Figure 8.2 shows our semi-layered architecture for the framework. The leftmost part of the figure shows the layered view of the framework. Applications use the interface provided by the Frameworks core functionality. The Framework uses a generic interface to control technology specific Network modules. This technology specific network module can be implemented with technologies such as Bluetooth, ZigBee or WLAN. This module layer along with the Network interface layer, is what makes the framework independent of the chosen network technology. The bottom layer is J2ME itself along with the specific network technology APIs. The rightmost part of the figure labelled "Domain" contains the abstractions of the domain concepts Node, Group, Service and Message. The boxes above the one labelled "J2ME + Network specific APIs" will all be realized as separate packages in our more detailed design.

8.3 Design Changes

We will in this section document the design changes that have been done during this project.

8.3.1 Changes Caused by Problems in Original Design

The prototype developed in 2004 [37], uncovered some shortcomings in our original design. During our work on this project, we have redesigned some parts of the Peer2Me framework to improve these shortcomings. The following problems have been fixed:

Failing to connect more than two phones: In February 2005, we received an extra mobile phone for testing the applications we wrote for the Peer2Me framework. Tests uncovered that we

were unable to connect more than two phones in a network. We found that we had misunderstood certain aspects related to the way Bluetooth works and PANs work in general. The device listening for incoming connections is not functioning as a server, but as a slave device. The device connected to multiple other devices is the the one searching for other devices. We assumed a client-server model, when we in fact were working with a master-slave model.

Because of this, we had to reverse our approach to creating the networks with more than two devices. In order to fix this problem, the search and discovery process along with the handshake protocol were redesigned.

MIDlets freezing when initiating search: One quite large problem we encountered in [37] was that the example MIDlets we had created would freeze as soon as the user had to push a button to trigger an event. Later we discovered that this was because handling user input should be done as fast as possible. Method calls that take a long time to execute should be placed in a different execution thread than the event handling. Long event handling time would cause the event queue to be filled up and freeze the MIDlet's main execution thread.

Because of this problem, we had to introduce a new execution thread for searching for available devices.

Sending of messages as a potential block in MIDlet flow: This problem is closely related to problem of the MIDlets freezing when initiating search. We chose to not take any chances with this and introduced a message queue and a thread responsible for handling the message queue. The send message primitive would then only place a message ready for sending into the queue and return.

Problems catching exceptions in multi threaded environment: In order to deliver error messages to the actual MIDlet when an exception occurs, we had to introduce an interface for relaying the exception after it had been handled and wrapped in one of our own exceptions.

Without this interface we did not have any way of presenting the error message to the user, when for instance the process of sending a message failed.

8.3.2 Changes Caused by New Requirements

All the scenarios we worked with required persistence of data between sessions. Although Peer2Me was meant to be a framework for ad hoc networks, we decided to include a requirement for persistence manager to make it easier to create MIDlets that required to store and retrieve data. These two requirements, described in Chapter 7, introduced the concept of persistent objects and a persistence manager into the system.

8.4 Peer2Me Design

We will know document the Peer2Me design package by package. A logical architectural view, showing the main packages, is shown in Figure 8.3. All the packages, except the util package, in the Peer2Me framework can be related to the layered architectural overview shown in Figure 8.2.

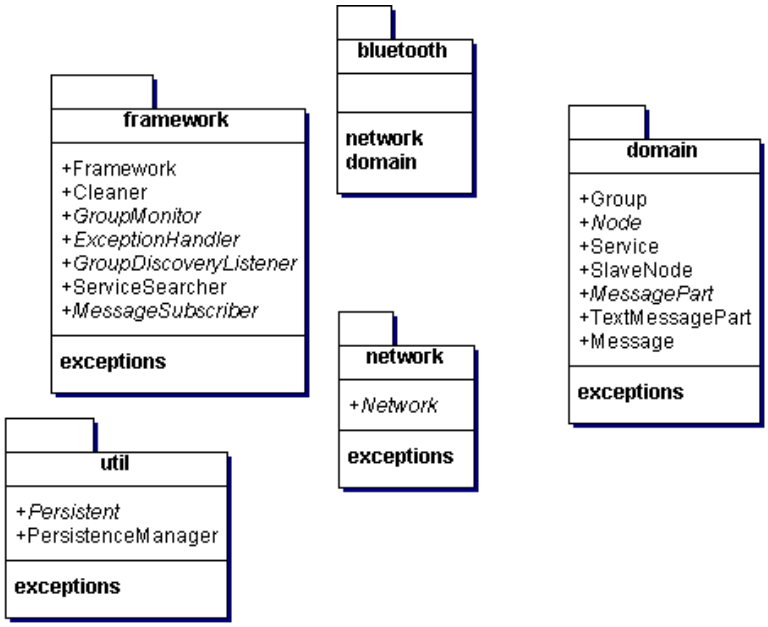


Figure 8.3: A logical architectural view showing the main packages.

The domain package relates directly to the box labelled domain in the overview, likewise does the framework package relate to the box labelled framework. The network package contains the network interface that the framework will communicate with the network module through. The specific network modules will each have their own package in the Peer2Me package hierarchy. A network module reference implementation using Bluetooth technology will be described in Section 8.5.

We will now give a more thorough description of all the packages in the Peer2Me framework. Each package is presented with a Unified Modeling Language (UML) diagram and a short description of each of its classes.

8.4.1 The Domain Package

The Domain package provides abstractions for a lot of the concepts presented in Chapter 8.1. Through the classes in this package, the applications can maintain a view of their environment.

Figure 8.4, shows the UML class diagrams of the classes and interfaces in the domain package. All classes will now be described individually.

Service

A *Service* object holds information about the service that the applications using the framework provide. Applications having registered the same service can find each other when they get within reach and communicate. An application should always register a Service object with its Framework instance.

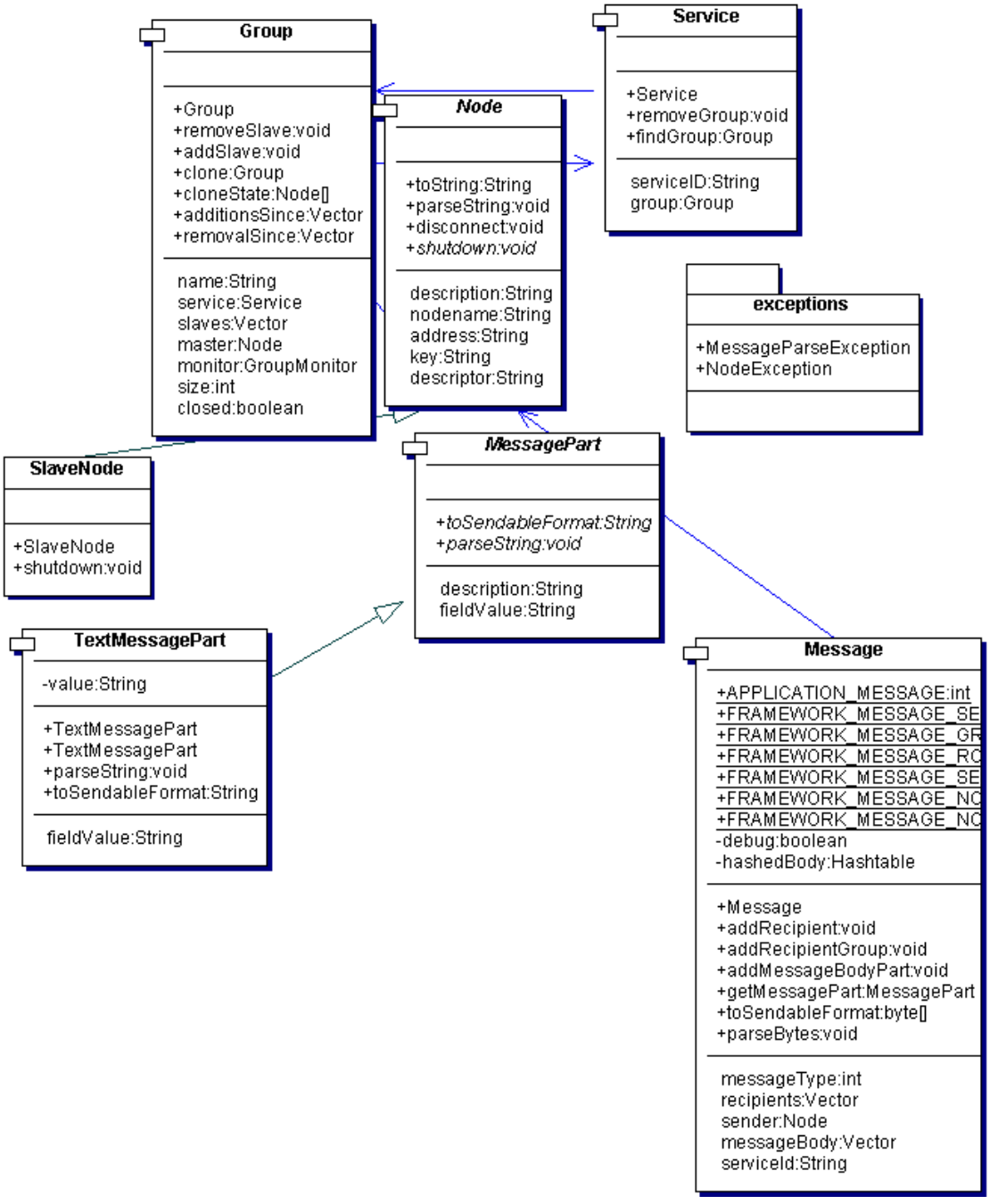


Figure 8.4: The UML diagram for the domain package.

Node

A device running an application using the framework is represented as a *Node*. The Node class is abstract, thus hiding network specific properties from the application and enabling it to focus utterly on communicating with other applications.

8.4.2 SlaveNode

SlaveNode is a subclass of *Node* and is used for representing nodes that do not have any network specific properties.

Group

A collection of connected nodes running the same service is represented as a group. The *Group* class provides functionality for administering members of a group, and sending messages to parts of or the whole group.

Message

The *Message* class encapsulates the messages sent between nodes in the network and provides methods for transforming the messages to a sendable format and back. A *Message* object can contain multiple parts, each associated with a key and thus giving us the means to retrieve a specified body part directly. There are some different kind of messages utilized by the framework. As seen in Figure 8.4, the different message types are identified with an integer identifier. The different message types along with their identifiers and main purposes are given in Table 8.1. A major part of the messages are utilized by special Peer2Me protocols described later in this chapter in Section 8.7.

MessagePart

A *MessagePart* is one of the atomic parts of the message body. The MessagePart class is abstract, ensuring that all message parts can be treated alike.

TextMessagePart

As of now, *TextMessagePart* is the only subclass of *MessagePart* incorporated in our design. TextMessagePart provides the application with the means of sending messages consisting of text only to other nodes.

Identifier:	Name:	Purpose:
1	APPLICATION_MESSAGE	This is a normal message, holding data specific to an application, sent between two applications on two different nodes.
2	SERVICE_INQUIRY	This message is used by the handshake protocol to find out if a node runs a given service. It contains a service identifier.
3	GROUP_DESCRIPTION	This message is used by the handshake protocol. When a node joins a group it receives a message that holds a description of the group.
4	ROUTE_MESSAGE	This is a normal application message, but its marked as a route message to ensure that the master of the group routes it to the given recipients.
5	SERVICE_ACK	This message is used by the handshake protocol to acknowledge that a node runs a given service.
6	NODE_JOIN	This message is used by the handshake protocol to give information to nodes that another node has joined their group.
7	NODE_LEFT	This message is used by the disconnection protocol to give information to nodes that a node has left the group.

Table 8.1: The different message types used by the framework.

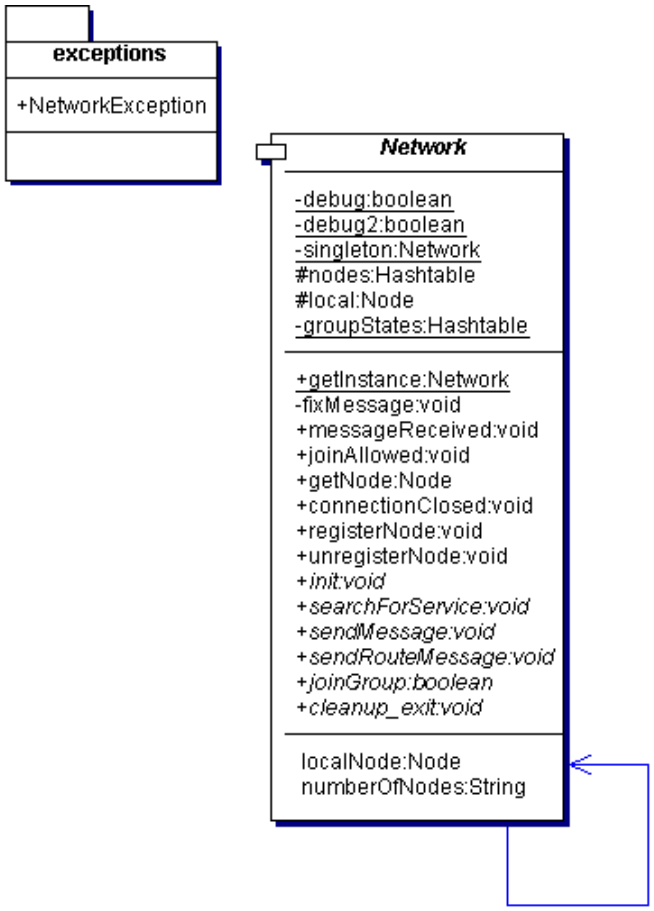


Figure 8.5: The UML diagram for the network package.

8.4.3 The Network Package

The Network package provides the services and primitives required by the framework to run independently of a network implementation.

Figure 8.5, shows the UML class diagrams of the classes and interfaces in the network package. This package contains only one class, which will be described here.

Network

The *Network* class is an abstract class giving the other framework modules an interface to the network layer. The implementation of a *Network* subclass will function as a central entity in new network modules.

We have included a design of our Bluetooth network module in Section 8.5 for further elaboration on how to implement different network transport mediums.

8.4.4 The Framework Package

The Framework package contains the application's main interface for controlling the functionality of the framework. The framework is instanciated, initialized and controlled through the class *Framework*. The interfaces *MessageSubscriber* and *GroupMonitor* are used by the framework to deliver information and commands to the application during runtime.

Figure 8.6, shows the UML class diagrams of the classes and interfaces in the framework package. All classes will now be described individually.

Framework

The *Framework* class, as mentioned earlier, is the main controller of the framework. Through the primitives in this class, the applications can manipulate and use the framework functionality providing the application with abstractions from the network layer when it comes to groupware functionality.

GroupDiscoveryListener

The *GroupDiscoveryListener* interface is implemented by all MIDlets running as slaves in a network. When a master initiates a search for other nodes running the same service and finds the slave node, the slave node is notified of the master's discovery. The slave node is then given the option to join the master's group.

GroupMonitor

The *GroupMonitor* interface provides an application with the means to control a group. By implementing this interface, an object can be asked to decide if a new node should be allowed to join the group or not. The GroupMonitor object will also be notified when nodes join or leave a group.

MessageSubscriber

The *MessageSubscriber* interface enables the application to subscribe to messages arriving at the local device. If a class implements this interface, it can be registered with the framework as a MessageSubscriber and thus be notified of incoming messages.

ExceptionHandler

The *ExceptionHandler* interface introduces a method that allows Exceptions thrown in other execution threads than the main MIDlet thread to be delivered to the application. By implementing this interface, the MIDlet developer can decide how error messages are displayed to the user.

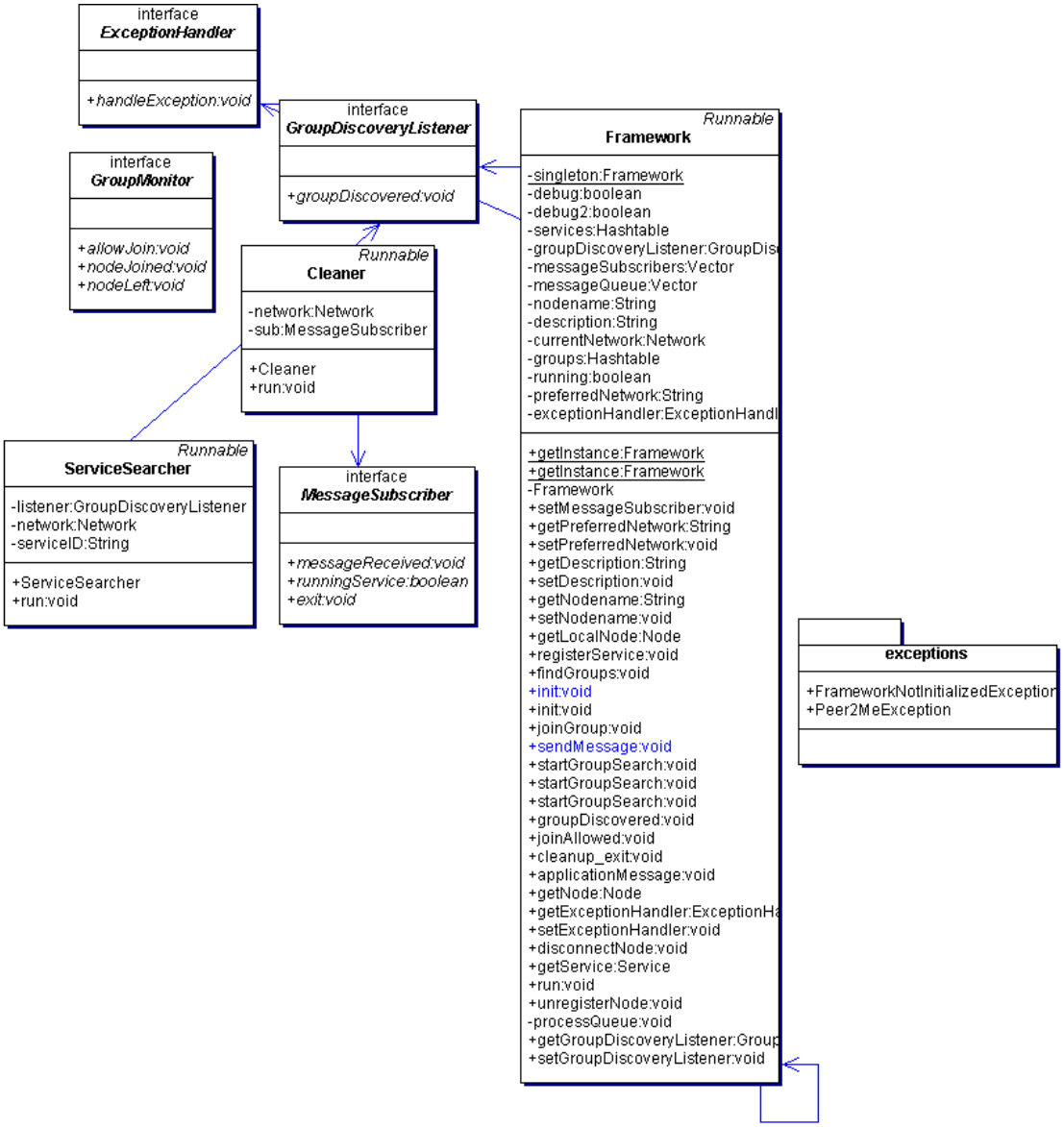


Figure 8.6: The UML diagram for the framework package.

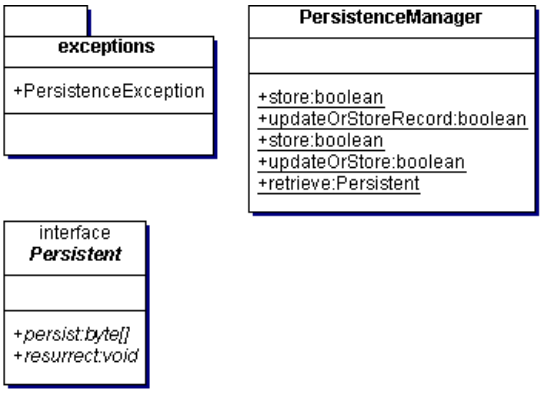


Figure 8.7: The UML diagram for the util package.

8.4.5 The Util Package

The Util package provides support functionality for the applications using the Peer2Me framework. Different kinds of utilities can be gathered in this package. The classes of the util package is shown in Figure 8.7. We will now describe each class individually.

Persistent

All objects that are to be persistent must implement the *Persistent* interface with its two methods *resurrect(byte[] data)* and *byte[] persist()*.

PersistenceManager

The *PersistenceManager* class is a wrapper class for the J2ME RecordStore, providing a key-oriented persistence layer the applications can use for storing objects between applications sessions. Through the *PersistenceManager* the application can store and retrieve objects that implement the *Persistent* interface.

8.4.6 Runtime Behaviour

This section describes selected aspects of Peer2Me's runtime behaviour. Peer2Me's runtime behaviour is quite complex which makes sequence diagrams grow large and unreadable. Because of this, we have chosen to use textual descriptions of Peer2Me's runtime behaviour instead. The only sequence diagram included is simplified to increase readability.

Initializing the Framework

Figure 8.8 shows the process of initializing the framework and thus making it ready for use. The application first uses the *getInstance* method for retrieving the framework instance (not shown in the figure). When the application has a reference to the framework instance, it is initialized through the *init* method. The init method then retrieves and initializes the network instance.

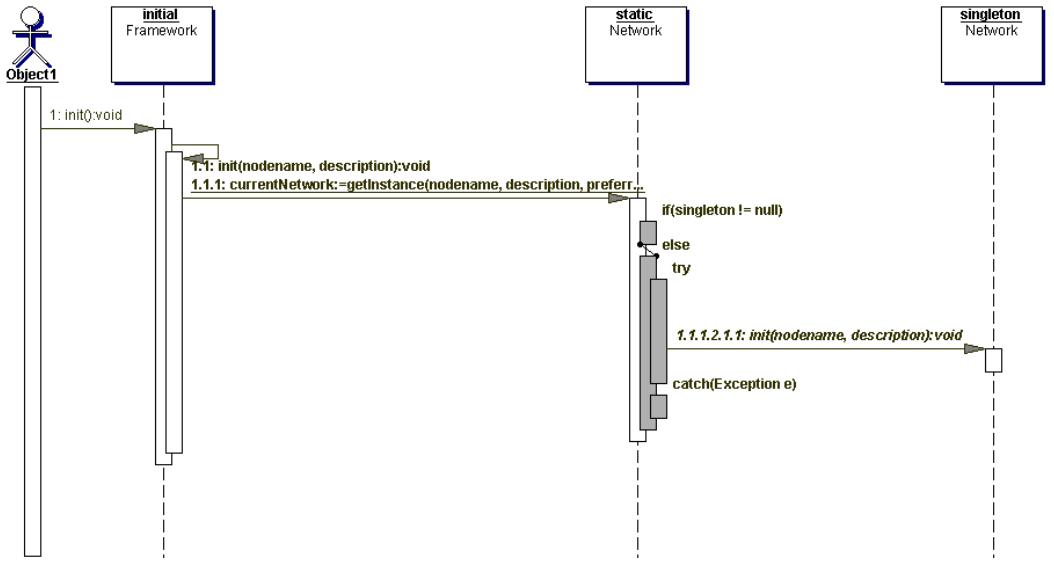


Figure 8.8: A sequence diagram showing the process of initializing the framework.

Sending a Message

The sending of messages in the Peer2Me framework needs to use a separate thread for the actual sending of the messages. This thread reads from a queue and sends the messages in the queue. If there are no messages in the queue, the thread sending the messages will pause and wait for the arrival of new messages.

Sending messages from the application perspective is done by handing over a message object to the framework. The framework then sets some required fields in the message on behalf of the application and places the message in the message queue to be processed by the sending thread.

This separate message sending thread is designed to ensure that processing events from the user interface can be done as quickly as possible. This, due to the fact that J2ME MIDlets have a tendency to freeze if user interface events are not processed fast enough.

Receiving a Message

When a node receives a message from one of the other nodes, this message is of one of two categories, messages meant for the framework instance, or messages meant for the application using the framework. The different types of messages need to be handled differently. In order to do this we have the method *messageReceived* in the *Network* class. Depending on the type of message (defined by the constants declared in the *Message* class) various actions should be taken.

Messages intended for the framework are handled and answered if required. These types of messages include messages in the different protocols and routing messages. Routing messages are a special case of application messages that needs to be routed through the master. The framework

handles the routing messages by resending them as application messages to all nodes in the recipient list. A more detailed description of the different protocols are given later in this chapter, in Section 8.7.

Messages intended for the applications are received and handed over to the application's registered message subscribers.

8.5 Bluetooth Module Design

This section presents the design of the Bluetooth specific parts of our system. These parts is found in the Bluetooth package, shown in Figure 8.2. The motivation for separating the actual network layer from the rest of the framework is first of all to make it easier for developers to migrate from one network medium to another. The separation of the framework functionality and the network implementation, gives this section two parallel goals. One goal is to present the actual design of the Bluetooth specific implementation. The other aim is to illustrate how a network technology implementation is done in order to function as a guide for developers wanting to add a new network technology package to the software suite.

The prototype of Peer2Me developed in 2004 described in [37], included a Bluetooth module that this design is based upon. We started out with the original design from [37] and let the design evolve through this project. For every iteration in our development process, we have revised and optimized the design. The changes that have occured during the project lifetime are documented in Section 8.5.2, and then the complete revised design is documented package by package. First in this section we will give a short overview of the Bluetooth protocol stack.

8.5.1 Bluetooth Protocol Stack

The Bluetooth specifications define the functionality of the Bluetooth protocol stack shown in Figure 8.9. In this project we will use a Java implementation of this protocol stack, called JSR-82. The JSR-82 specification, also known as the Java APIs for Bluetooth wireless technology (JABWT), is an optional package for J2ME defined and developed by the Java Community Process (JCP).

We will use the RFCOMM protocol that is as shown in Figure 8.9 situated on top of the L2CAP protocol. RFCOMM, also called the RS232 Serial Cable Emulation Profile, is a simulation of a serial port connection between two devices providing the programmer with input- and output-streams for communication. These streamconnections are reliable and bidirectional and are perfect for communication between nodes in the Peer2Me framework. The RFCOMM implementation in JSR-82 is based upon the classes StreamConnectionNotifier and StreamConnection defined in CLDC.

8.5.2 Design Changes

The following design changes have been made in the Bluetooth Network Module during this project:

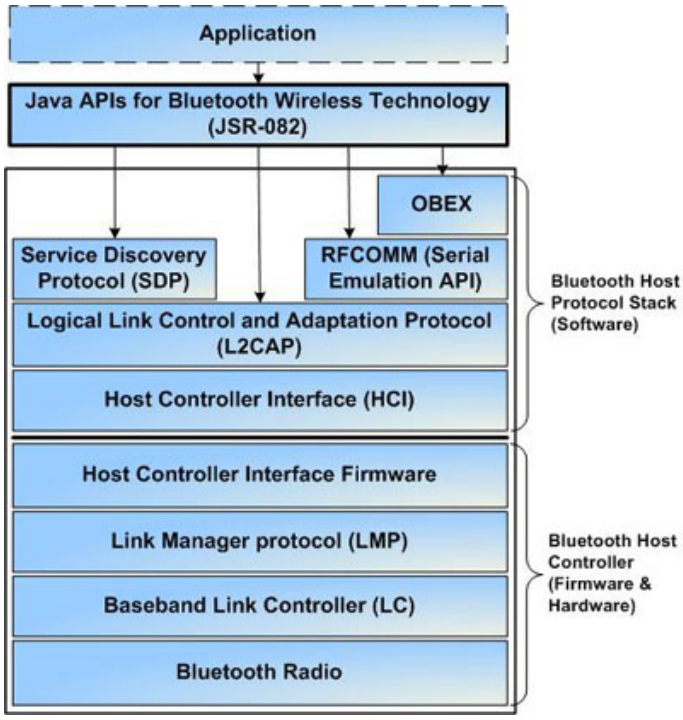


Figure 8.9: Overview of the Bluetooth protocol stack [45].

Introducing exceptions: In order to pass error messages from the Bluetooth module to the above layer we had to introduce our own exceptions in the module. In the Bluetooth Domain package we created a BluetoothNodeException that is thrown if something goes wrong at the node level. In the Bluetooth Network package we added a BluetoothNetworkException to handle errors or unexpected events in the that class. The BluetoothNodeException extends Nodeexception and the BluetoothNetworkException extends Networkexception. This use of inheritance ensures the same level of independence between the layers.

BluetoothNodelistener interface removed: We removed the BluetoothNodelistener interface simple because we did not want the BluetoothNetwork class to implement it. Removing this interface makes the BluetoothNetwork class and the ConnectionService class more tight coupled, but that does not matter because they are residing in the same package. Independency between any layers are not disturbed.

Fixed problems caused by interference with non-Peer2Me Bluetooth-enabled mobile phones: We have made some adjustments to the different classes to ignore all Bluetooth devices that are not running the Peer2Me framework, during the device discovery process. This makes the framework more reliable and speeds up the discovery process.

Generalization of network data and functions: During every iteration of the project we have tried to generalize as much code as possible from the specific Bluetooth module to the network interface package of Peer2Me or even further up in the architecture layers. It is very important that methods and/or attributes that not are specific for the Bluetooth-functionality resides in the network interface or further up. By moving functionality this way, making

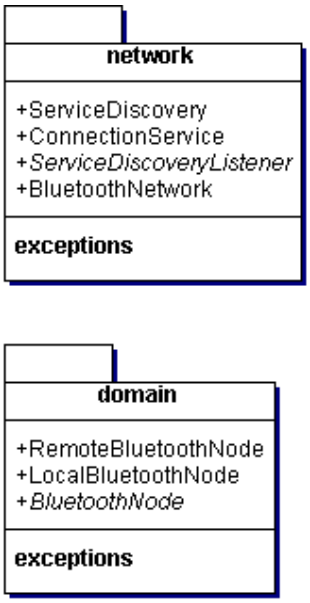


Figure 8.10: UML for the Bluetooth package.

the above layers more functional, less effort has to be spent when making another different specific network module (WLAN, ZigBee, IR,...).

Enabling parallel device and service discovery: When we started out this project, we started a parallel sub-project which aimed to optimize the Bluetooth module. The main bottleneck of the performance of Peer2Me is the slow process of device and service discovery offered by the Bluetooth module and hardware. We tried to optimize the design, mainly in the ServiceDiscovery-class, by making the discovery processes parallel. When one device is found during a device discovery, it should be possible to immediately start a service discovery on that device. This design tactic would dramatically reduce the response time of a complete device and service discovery. But unfortunately during implementation and testing of this new design we found out that the current version of the JABWT throws an exception when trying to do anything in parallel while discovering devices and services. Therefore we had to rewrite the discovery processes to be completely sequential and thereby slow the Bluetooth module down again. Although the process of discovery were made sequential again, we kept the infrastructure that makes parallelization possible. In this way, when some new version of the JABWT is released that offers parallel discovery processes, this can be easily integrated into the Bluetooth Module.

8.5.3 The Bluetooth Package

The Bluetooth package, shown in Figure 8.10, contains the Bluetooth implementations of the general domain and network packages. These packages are described in more detailed in the following sections.

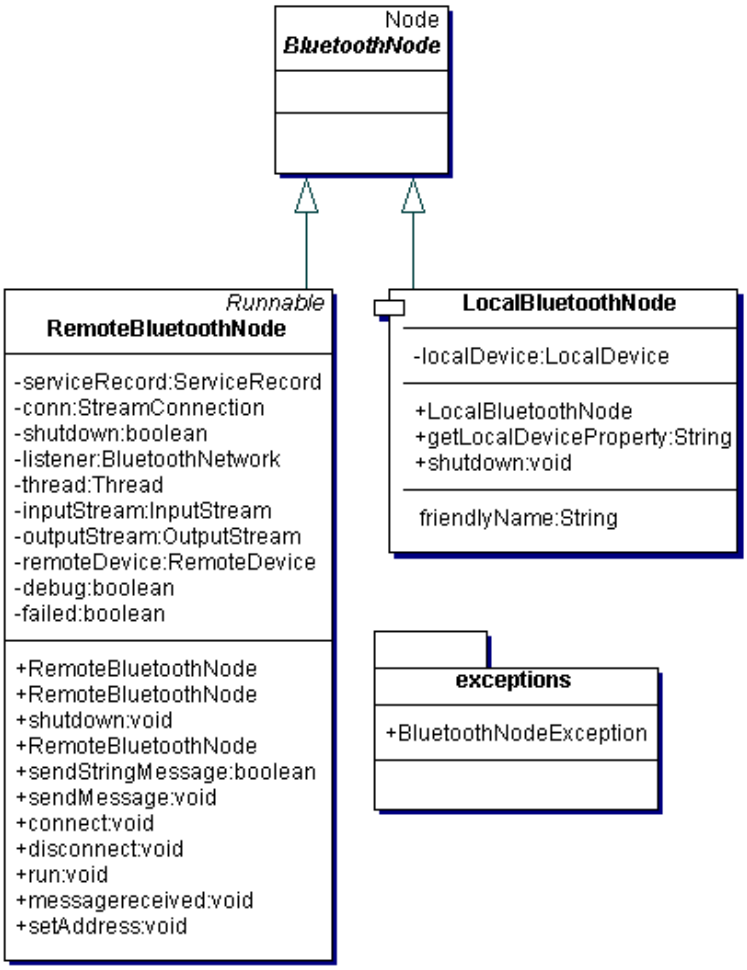


Figure 8.11: UML for Bluetooth Domain package.

8.5.4 The Domain Package

This package, shown in Figure 8.11, defines a Bluetooth implementation of the *Node* concept described in Section 8.1. A *BluetoothNode* has to subclasses called *RemoteBluetoothNode* and *LocalBluetoothNode*. RemoteBluetoothNode is a representation of another mobile device that is running the framework and a LocalBluetoothNode represents the local device.

BluetoothNode

A BluetoothNode is an abstract class that extends the Node class in the framework. It represents a general BluetoothNode, being a node that is using Bluetooth to communicate. There are two subclasses that inherits from this class, RemoteBluetoothNode and LocalBluetoothNode. These two subclasses only have one thing in common, their address, which is stored in the variable *blueToothAddress*.

LocalBluetoothNode

A LocalBluetootNode is a logical representation of a local Bluetooth device. The *BluetoothNetwork* class has a reference to an instance of this class. The BluetoothNetwork uses this class to obtain information and properties about the local Bluetooth device. A LocalBluetoothNode is created when the framework is initialized and associated with the already created BluetoothNetwork object.

RemoteBluetoothNode

A RemoteBluetoothNode is a logical representation of a remote Bluetooth device that is running the framework. The class contains different kinds of information regarding a remote node. It holds a connection to the remote node and a reference to a *ServiceRecord* object describing the service on the node. An *InputStream* and an *OutputStream* object are used for sending and receiving messages to and from the remote node. The BluetoothNetwork holds a list over all known RemoteBluetoothNodes. A RemoteBluetoothNode can be created by a BluetoothNetwork during service search or by a *ConnectionService* after the remote node has opened a connection. When an object of this class is created, an inputstream and an outputstream are opened. A RemoteBluetoothNode implements the *Runnable* interface, defined in CLDC 1.0, and the run method continously reads the inputstream and starts the parsing of incoming messages. When the parsing process is completed, it delivers a complete *Message* object to the BluetoothNetwork. Messages are sent to this node by calling the *sendMessage* method.

8.5.5 The Network Package

The Network Package defines an infrastructure for Bluetooth communication (see Figure 8.12). It holds basic network functionality like the creation of connections between devices, search for devices and services, and functionality for sending and receiving messages.

ServiceDiscovery

This class is responsible for doing the low level Bluetooth discovery operations. It is used by the BluetoothNetwork class to search for nearby devices running the framework. When a *ServiceDiscovery* object is created, a *ServiceDiscoveryListener* is associated with it. When a device that is running the framework is found, a *ServiceRecord* associated with this device is delivered to the ServiceDiscoveryListener. The ServiceDiscoveryListener, usually a BluetoothNetwork, can then create a RemoteBluetoohNode that represents this particular device that is running the framework and further ask this RemoteBluetoothNode if it is running a specific framework service.

The first step of the discovery process is to discover the nearby devices that are in range. A *DiscoveryAgent* class in the Bluetooth Java APIs provides the methods needed for device discovery. A method called *startInquiry* is used to start the search and for each device found the method *deviceDiscovered* is called.

If a device has the right Class Of Device (COD) code (0x200 for mobile phones), a service search is initialized on that device. This is a search for Bluetooth services and not framework services.

The goal of this process is to find those nearby mobile devices that are running the framework. The framework is registered as a Bluetooth service with a Universal Unique Identifier (UUID). This id is a 128-bit value that is unique across space and time. When a service search is completed the method *serviceSearchCompleted* is called. A vector containing the ServiceRecords found on the particular device is then delievered to the ServiceDiscoveryListener.

ServiceDiscoveryListener

A class that wants to do a ServiceDiscovery and receive callbacks from the ServiceDisocvery has to implement this interface. The class implementing this interface has to define two methods. A method for handling the completion of a service discovery and another method for dealing with errors.

BluetoothNetwork

In this class all main network operations are executed or initialized. BluetoothNetwork is the main class of the Bluetooth network package. The class inherits from the abstract class Network. This class holds a complete list of all known RemoteBluetoothNodes and a reference to the LocalBluetoothNode. The BluetoothNetwork is used by the framework to send messages, perform service searches, register nodes and monitoring and administering the status and threads specific to the Bluetooth module.

ConnectionService

When a BluetoothNetwork is initialized, a *ConnectionService* is also initialized. There is a one-to-one dependency between BluetoothNetwotk and ConnectionService. When a ConnectionService object is created, an RFCOMM server is initialized with a connection string based upon the framework UUID. This creation returns a *StreamConnectionNotifier* object. From here the ConnectionService runs continuously, using the StreamConnectionNotifer object to accept and open new connections. For each new connection a RemoteBluetoothNode is created. This RemoteBluetoothNode object is given a reference to the BluetoothNetwork and the newly accepted connection.

8.6 Patterns

Design patterns are known solutions to known problems. The introduction of patterns into the development cycle makes sure that we do not have to resolve problems were known solutions have been tried and tested previously. Patterns also make sure that the code's readability increases for new developers that examine, modify or extend the code.

In our project there are three patterns we see fit to use: The singleton pattern, the observer pattern and the publisher-subscriber pattern.

8.6.1 Singleton Pattern

The singleton pattern is thoroughly described by E. Gamma et.al. in [23]. According to Gamma, the singleton pattern is used to ensure that a certain class will only be instanciated once. This limit is enforced by letting the class itself handle the instanciation process, thus giving it full control over its instances. Gamma also lists two major applications of the singleton pattern which are:

- There must be exactly one instance of a class, and it must be accessible to clients from a well-known access point.
- When the sole instance should be extensible by subclassing and clients should be able to used an extended instance without modifying their code.

In the Peer2ME framework, we will use the singleton pattern for two different classes: The *Network* class and the *Framework* class. Both are classes that never should be instanciated more than once. In addition, using the singleton pattern for loading the network instance will make it easier to fulfill the requirement of modular network support. This is according to the usage list provided above.

8.6.2 Observer Pattern

In [23], Gamma describes the Observer pattern. The observer pattern is used for notifying one or more objects of changes to a central entity. In Peer2Me a *Group* instance is mirrored on all the nodes in the group. Changes in a group are propagated through the network and the framework instance that receives a group change message changes its local copy of the group. When this object changes all consumers, such as the application, of the object needs to be notified. This is achieved through an observer pattern. This pattern will be realized through the *GroupMonitor* interface. An application and other users registers as GroupMonitors and are notified of all changes to the group.

8.6.3 Publisher-Subscriber Pattern

In [20], Edwards et.al. give a summary of the publisher-subscriber pattern. This pattern is in many ways similar to the observer pattern in the sense that they both handle events occurring at a central entity. The difference is that the observer pattern is used for notifying the observers of changes to one specific entity, whereas the publisher-subscriber is used for notifying the subscriber of the arrival of a new entity e.g. a message.

In the Peer2Me framework, we will use the publisher-subscriber pattern for three different areas:

- Delivering messages to the application.
- Raising exceptions to the application when something goes wrong.

In both cases, messages are handed over to the subscriber, the application, by the publisher, the framework. The first case deals with messages received from remote nodes and the second case deals with notifying the application of exceptions and errors that arise during runtime in the framework. The two cases will be realized by the *MessageSubscriber* and *ExceptionHandler* interfaces respectively.

8.7 Protocols

In the area of computer science, a protocol is a convention or standard that controls or enables the connection, communication, and data transfer between two computing endpoints. To design a framework that handles the creation of groups, joining of nodes and communication between these nodes we have to define some basic protocols that we will have implement. The three protocols described in this section provide basic functionality that a lot of other functionality is based upon. The protocols have evolved throughout the whole project to become mature, fault-tolerant and complete.

8.7.1 The Handshake Protocol

When devices connect as nodes in a group, certain messages need to be exchanged as a handshake. This handshake is done with 4 messages pr. node and complemented with up to 5 messages to existing members of the group. Figure 8.13 shows a graphical representation of the handshake protocol. The handshake goes as follows:

1. **Service Inquiry** When the master device discovers a slave, it sends a service inquiry message to check if the slave is running the desired service.
2. **Service Acknowledgement** If the slave is running the service, it responds with a service acknowledgment message.
3. **Group Description** If the group is defined as open, a group description containing all the nodes in the group is sent by the master to the slave upon receipt of the service acknowledgment. If the group is defined as closed, the application is asked whether or not the new node should be allowed to join before the group description is sent.
4. **Node Joined** When the slave receives a group description, it can join the group by sending a node joined message to master.
5. **Node Joined** After the master has received a node joined from a new node in the group, this join message is propagated to all the members of the group.

8.7.2 The Routing Protocol

When nodes that are not masters in a group send messages, the messages are always sent as route messages. The overall procedure for sending a message is:

1. The slave node sends a route message to the master node.
2. The master node examines the message and sends an application message to all the nodes in the recipient list. If the master node is included as a recipient, a message is also delivered to the application running on the master node.

If the node of origin is the master node, the messages will be sent as an application message immediately and the routing protocol will not be employed. Figure 8.14 shows a graphical representation of the route protocol.

8.7.3 The Disconnection Protocol

When a connection to another node goes down, this is detected by the framework and a disconnection protocol is used to handle the disconnection. How the different network nodes actually detects a disconnection can vary between different network technologies. A method called *disconnectNode*, situated in the *Framework* class, is called on the node where the disconnection is detected. The functionality of this method should conform to the pseudo code given in Listing 8.1. This pseudo code shows that a master of a group sends out a special kind of message, a node left message, to all the other slaves in a group when it detects a disconnection of a slave. In this way the master informs all the other slaves that another slave has disconnected. All the slave nodes that receive this message then removes the disconnected slave node from the group. The local node also informs all applications, running on the local node, that related to the disconnected node.

Listing 8.1: Pseudo code for the handling of disconnections of nodes.

```
1
2  For each group registered on the local node:
3
4      -If the local node is the master of the group:
5          -If the disconnected node where a slave in that group:
6              -Remove the node from the group
7              -Send node left message to all the other nodes in the group
8
9      -If the local node is not the master:
10         -If the disconnected node is the master of the group:
11             -Remove the group from the local node
```

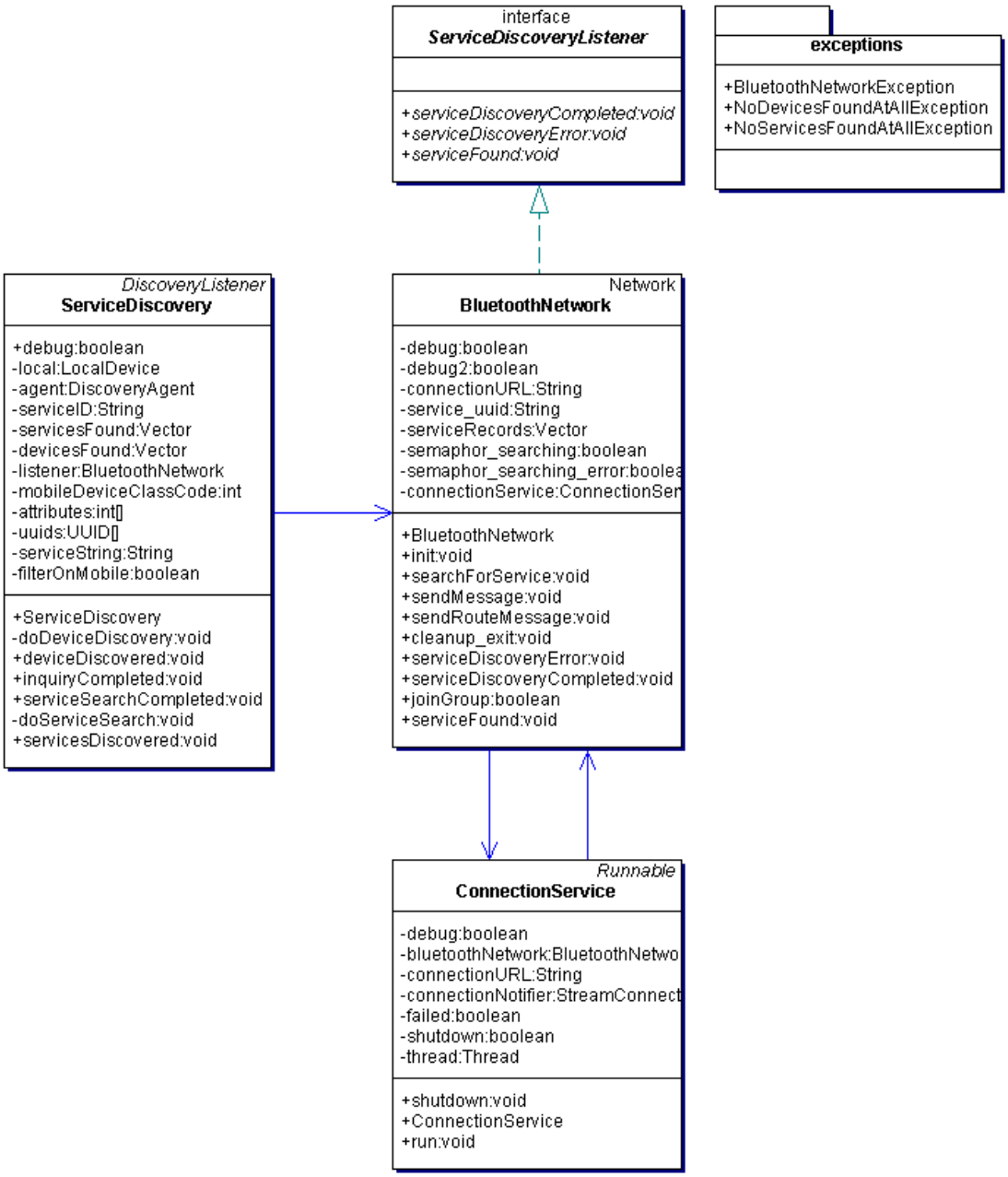


Figure 8.12: UML for the Bluetooth Network package.

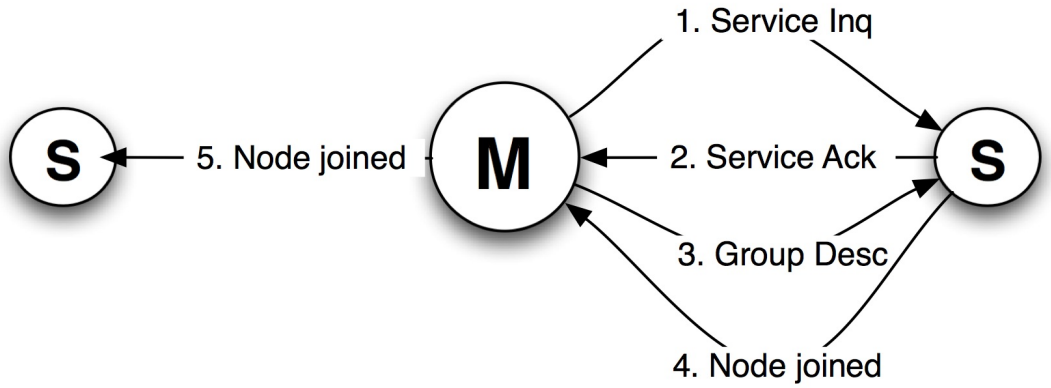


Figure 8.13: Messages in the handshake protocol.

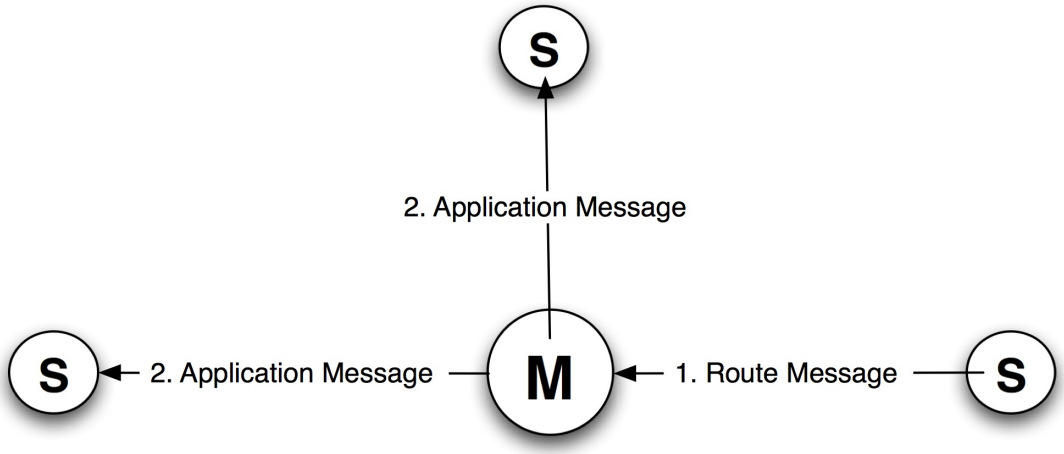


Figure 8.14: Messages in the routing protocol.

Chapter 9

Implementation

A lot of the code produced in the prototype described in [37] had to be completely rewritten for this project. As we have previously mentioned, we had misunderstood parts of the Bluetooth specification and made wrong assumptions of how the network topology worked. In addition, we have changed a lot of the code to make the prototype more robust. Robustness is a key feature on a framework running on ad hoc networks since disconnections happen often and the network is prone to experience errors. The complete source code from this project can be found on the attached CD or on our project website, http://www.peer2me.org.

9.1 Covered Functional Requirements

Table 9.1 gives an overview of the functional requirements coverage in the current version of Peer2Me. Requirements labelled with "Y" in the "Covered" column are fully covered while requirements labelled "P" are only partially covered. We will now give a more in depth description of each requirement's coverage:

FR1: Peer2Me is implemented using J2ME and JABWT and will run on any mobile phones with support for these APIs. This requirement is covered.

FR2: Peer2Me uses Bluetooth and creates wireless ad hoc Bluetooth networks. This requirement is covered.

FR3: When a new slave comes within range of an existing master, it can be included in the network when discovered by the master device. This requirement is covered.

FR4: All nodes in a group can send messages to each other by creating message objects and sending them through the framework instance. This requirement is covered.

FR5: When a master node searches for slaves it discovers all slaves within reach that are running the same application. The discovered slaves are then given the option to join if the master allows them to. This requirement is covered.

FR6: A message object can have one or more recipients in its recipient list. This requirement is covered.

FR7: Master devices can search for and connect to all slave devices that run the desired service. This requirement is covered.

FR8: Group objects have a property that specifies whether or not the group is closed or not. This property can be set at any time. This requirement is covered.

FR9: When a slave is discovered by a master administering a closed group, the application is asked whether or not the slave should be allowed to join or not. This requirement is covered.

FR10: See FR9. This requirement is covered.

FR11: When a slave is discovered by a master administering an open group the slave is automatically given the option of joining the group. This requirement is covered.

FR12: At the moment, there is no general mechanism for presenting decision messages, just simple information messages, to the user other than some specialized cases such as group administration. This requirement is therefore only partially covered.

FR13: Like FR12 there is no general way of delivering information notices to the framework other than the specialized cases when exception occurs, group changes or nodes are discovered. This requirement is therefore only partially covered.

FR14: The loading of the network module is completely dynamic, using the default J2ME class loader. At the moment, there is only a Bluetooth module available, but the framework is independent of the module to the degree that new modules can be implemented and loading into the framework without changing the framework code. This requirement is covered.

FR15: See FR14. This requirement is covered.

FR16: All messages are tagged with a sender address before they are sent from the originating node, thus ensuring that the sender of a message is known. This requirement is covered.

FR17: Objects can define how they are to be stored through the persistence interface. The persistence manager can then store persistent objects in J2ME's record store with an associated key. This requirement is covered.

FR18: Persistent objects that are stored in the record store can be retrieved via the persistence manager by their associated key. This requirement is covered.

A lot of the actual programming effort on the Peer2Me framework has been directed at quality aspects related to the non-functional requirements. Table 9.2 shows an overview of the non-functional requirements status, in the "Covered" column, "Y" means fully covered, "P" means partially covered and "N" means not covered. We will now give a more detailed description of how each non-functional requirement is covered in the Peer2Me implementation:

NFR1: When entering a message on one device and pressing the send button, it instantly appears on the receiving device's display. The actual sending of the messages is done in a dedicated thread always waiting for messages to send.

Requirement:	Requirement text:	Covered?
1	FR1: The system must support mobile phones.	Y
2	FR2: The system must support creation of ad hoc networks.	Y
3	FR3: The system must be able to connect to an existing ad hoc network.	Y
4	FR4: Nodes in a network must be able to exchange messages.	Y
5	FR5: The system must be able to create a group of nodes related to a specific application.	Y
6	FR6: The system must support multicasting and broadcasting of messages within a group.	Y
7	FR7: The system must be able to search for other phones supporting the same service	Y
8	FR8: The system must support the creation of groups as closed or open.	Y
9	FR9: The system must support to allow a node to try to join a closed group.	Y
10	FR10: The system must allow users in a closed group to reject other nodes to join the group.	Y
11	FR11: The system must support to allow a node to join an open group.	Y
12	FR12: The system must be able to present decision messages to the user.	P
13	FR13: The system must be able to present information to the user about framework related events.	P
14	FR14: The system must be able to support different kinds of network mediums.	Y
15	FR15: The applications must be independent of what network medium that is currently in use within the system. The application that is using the system to handle network traffic should not have to know what kind of network medium that is used by the device or make any kinds of adjustment to fit a specific network implementation.	Y
16	FR16: The system must be able to identify where a transfer originated from. To be able to send direct replies to a given device, it must be possible to see where a transfer originated from.	Y
17	FR17: The framework must include a mechanism for storing objects.	Y
18	FR18: The framework must include a mechanism for retrieving stored objects.	Y

Table 9.1: Covered functional requirements.

Requirement:	Requirement:	Covered:
1	NFR1: The framework must be able to transfer messages fast enough for real time interaction.	Y
2	NFR2: The framework must be able to detect the disconnection of nodes within a group and notify relevant applications and nodes about this.	Y
3	NFR3: The framework must adapt to disconnections and errors that arise due to the unstable nature of wireless networks.	P
4	NFR4: The framework must prevent applications from getting access to messages not addressed to them.	P

Table 9.2: Covered non-functional requirements.

NFR2: A node can detect and handle the disconnection of other nodes. A disconnection protocol has been implemented according to the design described in Chapter 8.7. Although this functionality is implemented and tested successfully on all available mobile phones, the time it takes for a node to detect the disconnection of another vary a lot between the different mobile phone manufacturers. This is because the Bluetooth functionality in the different mobile phones are not equally implemented.

NFR3: We have introduced a mechanism for notifying the framework and applications of errors and exceptions that occur during runtime. However, we have not yet completely covered all options when it comes to securing the framework from all possible failures or quality issues. For instance, if a message is sent from one node to another we have no mechanism for the acknowledgement of a received message. This implies that a node can not be 100% sure that a message that has been sent also has been received by the recipient. This requirement is therefore only partially covered.

NFR4: The routing protocol ensures that messages are never delivered to applications not in the recipient list. However, by implementing a hostile network module applications developers can listen to messages by intercepting them before they reach the framework instance. In order to protect messages from being read by unauthorized group members some kind of crypthographic strategy will have to be employed. Such a strategy is not implemented today, therefore this requirement is labelled as partially covered

9.2 Code Statistics

In Table 9.3 we list some statistical aspects related to the produced Peer2Me source code in order to illustrate the size of the Peer2Me framework in its current state. The column labelled “Value” contains the measurements of the framework specific code only. The column labelled “Including apps” contains measurement for all code produced for this project including all applications and test code. In Table 9.4 we have listed the size of the framework in a deployable format. The

Aspect:	Value:	Including apps:
Lines of code:	1163	2716
Number of classes:	26	71
Number of interfaces:	6	7
Number of packages:	18	29
Methods (avg. pr. class)		7.169
Maximum inheritance tree depth:	6	6

Table 9.3: Framework statistics.

Version:	Size of jar-file
Normal	40 KiloByte
Obfuscated	11 KiloByte

Table 9.4: Size of deployable framework jar-file.

table shows that the framework is more than small enough to be deployed on mobile phones. The obfuscated package jar-file of the framework is only 11 KiloBytes.

9.3 Code Examples

This section will present some code examples that illustrate some of the implementation techniques we have used to fulfill the design and the related requirements.

9.3.1 Loading the Network Module

One code example we would like to show from the Peer2Me framework is the loading of the network module. This is, as described in Chapter 8.6, done through a singleton pattern. In addition, the actual network implementation is loaded through the default J2ME class loader at runtime. Listing 9.1 shows the actual implementation of the singleton pattern. The *Network* class is an abstract class which serves as a superclass for all network modules. The static variable *singleton* holds a reference to the loaded network module. On line 9, the actual network module is dynamically loaded and instanciated.

Listing 9.1: Singleton pattern loads the network module.

```
1 private static Network singleton;
2
3 public static synchronized Network getInstance(String nodename, String
      description, String preferredNetwork){
4     if(singleton != null){
5         return singleton;
6     }
7
8     else{
9         try{
```

```
10
11          singleton = (Network)Class.forName(preferredNetwork).newInstance();
12          singleton.init(nodename, description);
13
14          }catch(Exception e){
15              ExceptionHandler eh = Framework.getInstance().getExceptionHandler()
                    ;
16              if(eh != null){
17                  eh.handleException(new Peer2MeException("Failed␣initializing␣
                        network␣module:␣" +                 e.getMessage()));
18              }
19          }
20          return singleton;
21      }
22  }
```

9.3.2 Sending Messages

The process of sending messages is of course an important part of the framework. We will now show how we have implemented one part of this functionality. The two methods, *sendMessage*, shown in Listing 9.2, and *processQueue*, shown in Listing 9.3, are responsible for handling outgoing messages on the framework level. They are both situated in the *Framework* class. The whole process starts with an application that calls the sendMessage method. The sendMessage method thereby decides if this message will have to be routed through the master or not. Then it adds some framework-specific information to the message and puts it in a queue. The synchronized method processQueue is then notified. The processQueue method implements a simple First In and First Out (FIFO) queue for application messages. This ensures that messages are sent correctly and that parallel messages from different applications are sent sequential. Messages are processed in two different ways in the processQueue method, as route messages or ordinary messages.

Listing 9.2: The method for initiating the sending of messages in the Framework class.

```
1
2   public synchronized void sendMessage(Message message, Service toService){
3
4       Group currentGroup = getService(toService.getServiceID()).getGroup();
5       if(currentGroup.getMaster() == getLocalNode() && message.getMessageType()
            == 0){
6           message.setMessageType(Message.APPLICATION_MESSAGE);
7       }else if(message.getMessageType() == 0){
8           message.setMessageType(Message.FRAMEWORK_MESSAGE_ROUTE_MESSAGE);
9       }
10
11      TextMessagePart text1 = new TextMessagePart();
12      text1.setDescription("service");
13      text1.setFieldValue(toService.getServiceID());
14
15      message.addMessageBodyPart(text1);
16
17      message.setServiceId(toService.getServiceID());
18      messageQueue.addElement(message);
19
20      notify();
```

```
21
22 }
```

Listing 9.3: The processing of the message queue.

```
1
2  private synchronized void processQueue(){
3      try{
4          if(messageQueue.size() == 0) wait();
5          Message m = (Message)messageQueue.firstElement();
6          messageQueue.removeElement(m);
7          if(m.getSender() == null){
8              m.setSender(currentNetwork.getLocalNode());
9          }
10
11         try {
12             if(m.getMessageType() == Message.APPLICATION_MESSAGE){
13             currentNetwork.sendMessage(m);
14             }
15
16             else{
17                 Node master = getService(m.getServiceId()).getGroup().getMaster
                       ();
18                 currentNetwork.sendRouteMessage(master, m);
19             }
20
21             } catch (NetworkException e1) {
22                 if(exceptionHandler != null)exceptionHandler.handleException(e1
                       );
23             }
24
25         }
26
27     catch(InterruptedException e){
28         if(exceptionHandler != null)exceptionHandler.handleException(e1);
29     }
30 }
```

9.3.3 Building Group Objects

The abstract *Network* class holds a major part of the functionality for receiving and processing incoming messages. Incoming messages are processed according to their message identifiers. This functionality is mainly implemented in a method called *messageReceived*. We will now explain an example of how this method handles incoming messages. As explained above, each different message type requires different processing and actions. We will now show how the Network class handles incoming messages of one special category, the GROUP_DESCRIPTION messages. This message is part of the handshake protocol, described in Chapter 8.7. When a node joins a group, it receives a message from the master of the group that holds a description of the group. The code in Listing 9.4 shows how the incoming message is processed. The message holds information about the group and this information is then used to create a group object reflecting this information. The master is set and the different slaves in the group are created and registered. Finally, when the group object is completely created, it is delivered to the *Framework* class that notifies and delivers

the group object further to the right application. The application can then handle the group object as it wants, for instance sending a message to one of the nodes in the group.

Listing 9.4: The making and delivering of a group object from a description message.

```
1
2  Group groupFound = new Group();
3  MessagePart m;
4  Service service = Framework.getInstance().getService(message.getMessagePart("
       service").getFieldValue());
5
6  if(service != null){
7      m = message.getMessagePart("group");
8      groupFound.setService(service);
9      groupFound.setName(m.getFieldValue());
10     String master = message.getMessagePart("master").getFieldValue();
11     Node masterNode = getNode(master);
12     groupFound.setMaster(masterNode);
13
14     String slave;
15     int slaveId = 0;
16
17     while((m = message.getMessagePart("slave" + slaveId)) != null){
18         slave = m.getFieldValue();
19         Node slaveNode = getNode(slave);
20         if(slaveNode == null){
21             slaveNode = new SlaveNode();
22             slaveNode.setAddress(slave);
23             registerNode(slaveNode);
24         }
25         slaveId++;
26     }
27
28     Framework.getInstance().groupDiscovered(groupFound);
29
30 }
31
32 else{
33     ExceptionHandler eh = Framework.getInstance().getExceptionHandler();
34     if(eh != null){
35         eh.handleException(new NetworkException("Received␣groupdescription␣for␣
               unknown␣unavailable␣service:␣" + message.getMessagePart("service").
               getFieldValue()));
36     }
37 }
```

9.3.4 Handling Bluetooth Service Search Results

We also want to show one code example from the Bluetooth network module. As explained in the Bluetooth module design chapter, Chapter 8.5, the Bluetooth module is based upon the Java APIs for Bluetooth Wireless Technology (JABWT). Listing 9.5 shows a part of the *ServiceDiscovery* class of the Bluetooth network module. This class is responsible for searching after other nodes (mobile phones) that support Bluetooth and that run the Peer2Me framework. The code in the listing shows a part of the class that is responsible for finding all nodes that actually run the

Peer2Me framework. This search is done among all the nodes that have been found supporting Bluetooth. When the search is finished, all the nodes in the result set are then handed over to the *BluetoothNetwork* class by using a listener interface. The BluetoothNetwork class then further establishes connections to all the different nodes for further communication.

Listing 9.5: Handling Bluetooth service search results.

```
1
2  if(devicesFound.size()>0){
3      try{
4          agent.searchServices(attributes,uuids,(RemoteDevice)devicesFound.
               firstElement(),this);
5          devicesFound.removeElementAt(0);
6
7      }
8      catch (BluetoothStateException e) {
9          ExceptionHandler eh = Framework.getInstance().getExceptionHandler();
10         if(eh != null){
11             eh.handleException(new BluetoothNetworkException("Could␣not␣start␣
                   service␣␣␣␣␣␣␣␣␣␣␣search"+e.getMessage()));
12         }
13     }
14 }
15
16 else{
17     if(servicesFound.size()==0){
18         ExceptionHandler eh = Framework.getInstance().getExceptionHandler();
19         if(eh != null){
20             eh.handleException(new NoServicesFoundAtAllException("No␣services␣
                   found␣at␣all:␣␣␣␣␣␣␣␣␣␣␣␣␣"+servicesFound.size()+"\n"));
21         }
22     }
23
24     else{
25         for(int i=0;i<servicesFound.size();i++){
26         listener.serviceFound((ServiceRecord)servicesFound.elementAt(i),
               serviceID);
27     }
28 }
```

Part III

The Peer2Me Applications

Chapter 10

Overview

This part of the report will present and describe the applications we have developed using the Peer2Me framework. The purpose of making example applications is manifold. As described in our engineering approach in Chapter 2.2.1, we developed applications in parallel with the framework to easily and quickly test, discover and add new necessary functionality. The example applications will be used both for testing the usefulness of the framework itself, but also the usability and benefit for users of such collaborative applications deployed in real-life scenarios.

When choosing what kind of example applications we wanted to develop, it was important that these applications would differ so much that they together would cover the most important aspects of the framework and the possible usage scenarios. Thus, we decided to make one application from each main category from the classification matrix described in Chapter 4.1.1. The classification matrix together with the chosen applications are shown in Table 10.1.

We will now give a short description of the three example applications:

Business Card Exchange: This application provides users with functionality for creating and storing business cards. These business cards can be made available and exchanged with other users. This application is placed in the "Hybrid" category because users can download a business card from a mobile phone without notifying or trigging any interaction from the owner of the phone. This means that the application handles user requests automatically, which places the application in the "Hybrid" category. The scenario that this application is based upon was not included in our last report, [37], but was described by Sveen and Kirkhus in [33]. This application was chosen because it illustrates the concepts of a "Hybrid" application in a simple way.

PAN Instant Messaging: Personal Area Network (PAN) Instant Messaging allows users to create and connect to small groups. Inside these groups the users can communicate with each other through text messages. All communication and interaction in this application is explicitly triggered by the users and the application is therefore placed in the "User" category. This application is based upon a scenario described in our previous report [37]. This application was chosen as an example application because it involves communication between a group of many nodes and utilizes all aspects of routing and group dynamics.

User Interaction/ Advanced P2P functionality	User	Auto	Hybrid
Multi-hop			
Single-hop	PAN Instant Messaging	Converging Top Ten List	Business Card Exchange

Table 10.1: Example applications placed in th Classification Matrix.

Converging Top Ten List: This application allows users to create and maintain top ten lists of prices of services or products. The application will automatically send these lists to other devices. The lists will be compared, calculated, updated and thereby converge towards the true top ten list. All the communication between phones is done automatically by the application. This places the application in the "Auto" category. This application is based upon a scenario described in our previous report [37]. This application was chosen because it illustrates the idea of automatic exchange of information very well and because it has a potential to help participants in the consumer market to find the best and cheapest products and services.

These applications will be further described in the following chapters of this part of the report.

10.1 Design Overview

In this section we will explain the general and common design principles of the three example Peer2Me applications. All three applications are based on the Model-View-Controller (MVC) architectural pattern. This pattern was invented by Trygve Reenskaug in the late 70s while working with the object oriented language Smalltalk. As stated in [47], MVC was conceived as a general solution to the problem of users controlling a large and complex data set. The main idea behind the pattern is to separate responsibility in an application and thereby classify objects into three modules:

Model: This is a representation and encapsulation of the real world data on which the application operates.

View: This presents the data from the model to the user using some kind of interaction technique.

Controller: This module responds to all kinds of events, including user interaction, and then invoke changes in the model, the view or both.

This way of designing applications ensures maintainability and reusability and makes the applications more scalable. For instance, it is easy to add a new and different view without changing the model and the logic of the application can be changed in the controller without affecting the view or the model. A graphical representation of these concepts and their relations is shown in Figure 10.1.

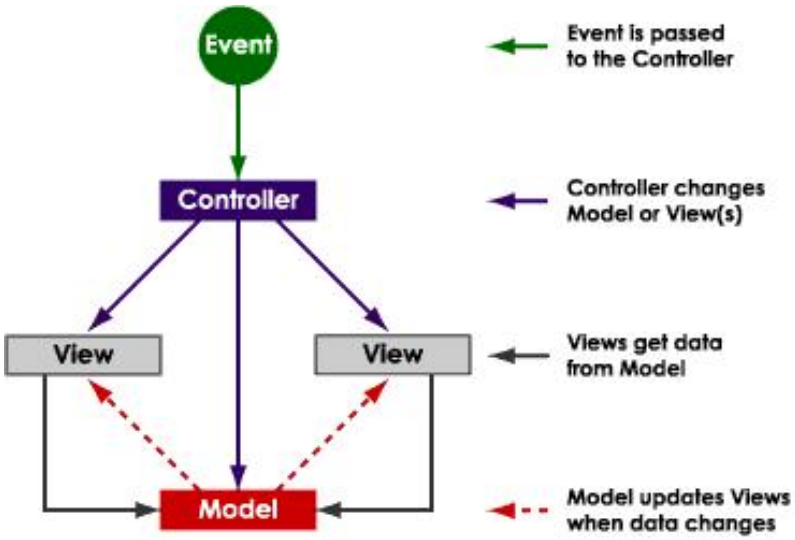


Figure 10.1: The Model View Controller pattern [7].

For each Peer2Me application, there is a package of classes holding the view part of the application and another package holding the model. The view part is mainly classes that utilizes the GUI functionality of MIDP 2.0 and presents information to the user by using lists, forms and other graphical objects. These views are also functioning as event listeners. When a user interacts with a view, the view will notify the controller about the event and the controller will then decide on further action. The controller of each application is a class that holds all Peer2Me specific code, for instance, joining nodes, sending messages and handling exceptions. In this way, the controller handles all the events that relates to the application, by handling events both from inside and outside of the application. The controller inherits from the MIDP 2.0 MIDlet class which means that it implements the methods *startApp*, *pauseApp* and *destroyApp* that characterizes a MIDlet.

10.2 Development and Testing

Developing mobile applications is a quite tedious process. To test the applications on the mobile phones in their real environments, they first have to be deployed on the phones and thereby tested in surroundings that may involve slow and unstable network connections. In Chapter 2.2.1, we described the overall development life cycle model for the framework and its applications. Figure 10.2 shows a more detailed process model of how we have worked when developing the three example applications. This model involves two different levels of testing activities. When we started developing applications we soon realized that it was far too time consuming to deploy and test the applications on the phones for each iteration. We therefore iterated a few times (design, implementation, emulator testing and evaluation) without testing the applications on the real phones, just using the emulator for testing. When a specific part or functionality was completed and ran perfectly on the emulator we deployed the code on the phones and tested it. We tested each deployment on the three different test mobile phones described in Chapter 2.3. The deployment of an application on a phone involves a few different steps:

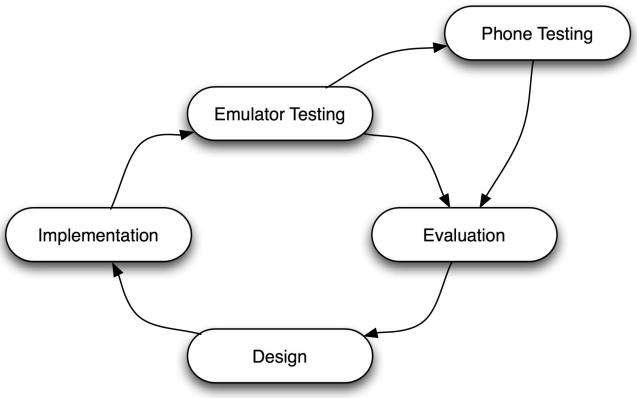


Figure 10.2: The application development model.

1. Compiling the code
2. Packaging of the code into a executable jar-file
3. Sending the jar-file using Bluetooth to the phones
4. Installing the jar-file on the phones
5. Running the jar-file on the phones

The testing on the phones themselves often revealed problems. This is first of all because an emulator runs the software in a totally different environment. The emulator has almost infinite memory, much more CPU power and the network connections are simulated. Another problem is that the different mobile phone manufacturers have implemented the CLDC, MIDP and the Bluetooth API functionality in slightly different ways. After testing an application on the phones, we went back to the evaluation phase and continued iterating to fix the problems that were revealed on the phones or to add new functionality.

When using an emulator, communication between several phones could be tested easily with just starting several instances of the emulator. We used the Wireless Toolkit 2.2 from Sun for emulation. Figure 10.3 shows an example running two instances of the emulator for testing an application.

Figure 10.3: Testing an application on the Wireless Toolkit 2.2.

Chapter 11

Business Card Exchange

This scenario was included in our previous report [37] and originally created by Sveen and Kirkhus in their master thesis [33], focusing on scenario analysis to find requirements for the framework as a whole. In this chapter, we will carry out a scenario analysis where the main focus will be on the user's perspective trying to uncover the requirements for this application only. After elaborating the requirements for the application, an overview of the design will be given, followed by a description of the implementation details of the application.

11.1 Scenario

The scenario takes place at a conference where two participants find that they have common research interests and want to meet again later in order to continue a discussion.

11.1.1 Goals and Preconditions

The preconditions of this scenario include that the actors each own a mobile phone supporting J2ME applications with MIDP 2.0 and some transfer medium such as ZigBee, Bluetooth or WLAN.

The goal of the scenario is to have two users exchange their electronic business cards and store them on their mobile phones. This can be stated as:

Goal 1: The application transmits an electronic business card to another phone.

Goal 2: The application stores all received business cards locally.

11.1.2 Normal Action Sequence

Mr. Smith is going to a conference on frameworks for creating web-portals. He thinks about the previous conferences he has attended and about how many people with similar interests as himself

he has met there. This time he would like to get electronic business cards from the people he meets. Before he leaves home, he downloads a new Application called Peer2Me BC-exchange and installs it on his mobile phone.

On his way to the conference Mr. Smith starts his new application and is automatically prompted to enter his own contact information. He enters his name, his phone number, his email address and the name of his company. He then gets a message saying he is ready to exchange his new electronic business card with others. Mr. Smith puts away his phone and hopes that the other participants have been smart enough to install the same application as well.

At the end of the last conference day, Mr. Smith is having a lively discussion about the future of web-portal development with his new found friend Mr. Jones. Unfortunately they both have planes to catch in order to get home, but they both want to continue their discussion at a later time. Mr. Smith and Mr. Jones both start their Peer2Me BC-exchange applications and Mr. Smith starts a search for other people offering their electronic business cards. He gets a list of available business cards to download and selects Mr. Jones'. When Mr. Smith downloads Mr. Jones' card, Mr Smith's card is also uploaded to Mr. Jones' phone. When the process is completed they both leave to catch their transport home, knowing they have each other business cards and will be able to get in touch to continue their discussion.

On his way to the airport Mr. Smith reviews all his downloaded electronic business cards by flipping through them on his phone.

11.1.3 Critical Exceptions and Error Checking

When exchanging business cards, the transfer of a business card may fail due to unknown circumstances such as the phones getting to far apart, disturbances in the air or one of the phones going out of battery. In cases where the transmission fails, it should be retried after prompting the user.

If a partial business card is received, the card should be discarded and the user notified of the transmission failure.

11.2 Requirements

We will now use the method described in Chapter 2 to analyze the scenario in order to uncover the functional requirements for the application.

11.2.1 Goal Analysis

We will now perform the goal analysis on each of the two goals stated above.

Question 1

Both the stated goals require computerized support in order to be achieved. This leads to two functional requirements stating:

FR1: The application must be able to send the users electronic business cards.

FR2: The application must be able to store the received business cards permanently.

Question 2

None of the goals describe any quality or performance properties.

Question 3

This question does not apply to any of the goals stated in the scenario.

Question 4

None of the goals require any management decisions about resources or responsibilities.

Question 5

None of the goals can be fully automated, this leads to a new functional requirement:

FR3: The exchange of business cards must be triggered by a user.

11.2.2 Inbound Event Analysis

In the scenario we can identify the following events:

In 1: Mr. Smith enters his own contact information into the phone the first time the application is started.

In 2: Mr. Smith searches for available business cards to download.

In 3: Mr. Smith chooses Mr. Jones' phone from a list of devices offering a business card and exchanges cards with Mr. Jones.

Requirements Elaboration

The inbound events result in the following requirements.

FR4: The application must prompt the user for his contact information the first time it is started. (From In 1)

FR5: The application must store the users contact information for later use. (From In 1)

FR6: The application must support searching for other people offering their business cards for download. (From In 2)

FR7: After choosing a device to exchange cards with, the card is stored as well as the sending the local card to the remote phone. (From in 3)

11.2.3 Categorize System Output

The following outputs from the system has been identified:

Out 1: When a search for available business cards finishes, Mr. Smith is presented with a list of the result.

Out 2: Mr. Smith can view each business card stored on his phone.

Requirements Elaboration

These outputs give us the following the functional requirements:

FR8: After searching for available business cards for download, the application must present a list from which the user can choose the desired card. (From Out 1)

FR9: Stored business cards must be available for browsing. (From Out 2)

11.2.4 Summary

This gives us the following requirements for the application:

FR1: The application must be able to send the users electronic business cards.

FR2: The application must be able to store the received business cards permanently.

FR3: The exchange of business cards must be triggered by a user.

FR4: The application must prompt the user for his contact information the first time it is started.

FR5: The application must store the user's contact information for later use.

FR6: The application must support searching for other people offering their business cards for download.

FR7: After choosing a device to exchange cards with, the card is stored as well as sending the local card to the remote phone.

FR8: After searching for available business cards for download, the application must present a list from which the user can choose the desired card.

FR9: Stored business cards must be available for browsing.

11.3 Design

The Business Card Exchange (BCEX) MIDlet is designed according to a Model View Controller pattern trying to separate the different mechanisms in the application in order to achieve maintainability and reusability of code. The main package, businesscard, contains the controller, which is the actual MIDlet subclass, *BCEXMIDlet*. All communication with the framework is done in this class. Figure 11.1 shows the layout of the packages in the BCEX MIDlet along with the actual MIDlet class. All the subpackages will now be described individually.

11.3.1 The Model Package

The model package contains the data model used in the BCEX MIDlet. The model is based on the two persistent objects *BusinessCard* and *BusinessCardCollection*. A BusinessCard contains contact information for one person. BusinessCardCollection wraps the user's own business card and all the contacts he has collected into one object. Both objects implement the *Persistent* interface, allowing them to be handled by the persistence support in Peer2Me.

11.3.2 The View Package

The view packages contains all the different menus, lists and information that can be displayed to the user. No actual processing of events or data is conducted in any of the classes contained in this package. All user interaction is just caught and forwarded to the actual MIDlet for processing. The graphical user interface hierarchy is shown in Figure 11.3 and the actual classes are shown in Figure 11.4. Each of these classes will now be described. Since the classes in this package only function as information displays we will not describe their behaviour, but list their purpose.

RegistrationForm: The first view the user is presented with when starting the application for the first time. Used for setting the user profile.

MainForm: This view is the main view, when the application is in its idle state, the main view is displayed on screen with the standard set of commands.

SearchResultList: After initiating a search for other devices running the same service the search result view is displayed. As new devices are discovered, they are added to this list one by one. When the list is complete, the business cards from the different devices are available for download.

BusinessCardReceivedForm: This view is used for displaying downloaded or previously stored business cards.

LibraryList: This view displays a list of all the cards currently stored in the library. The user can choose from the list and view details about each card. Details are shown in the BusinessCardReceivedForm.

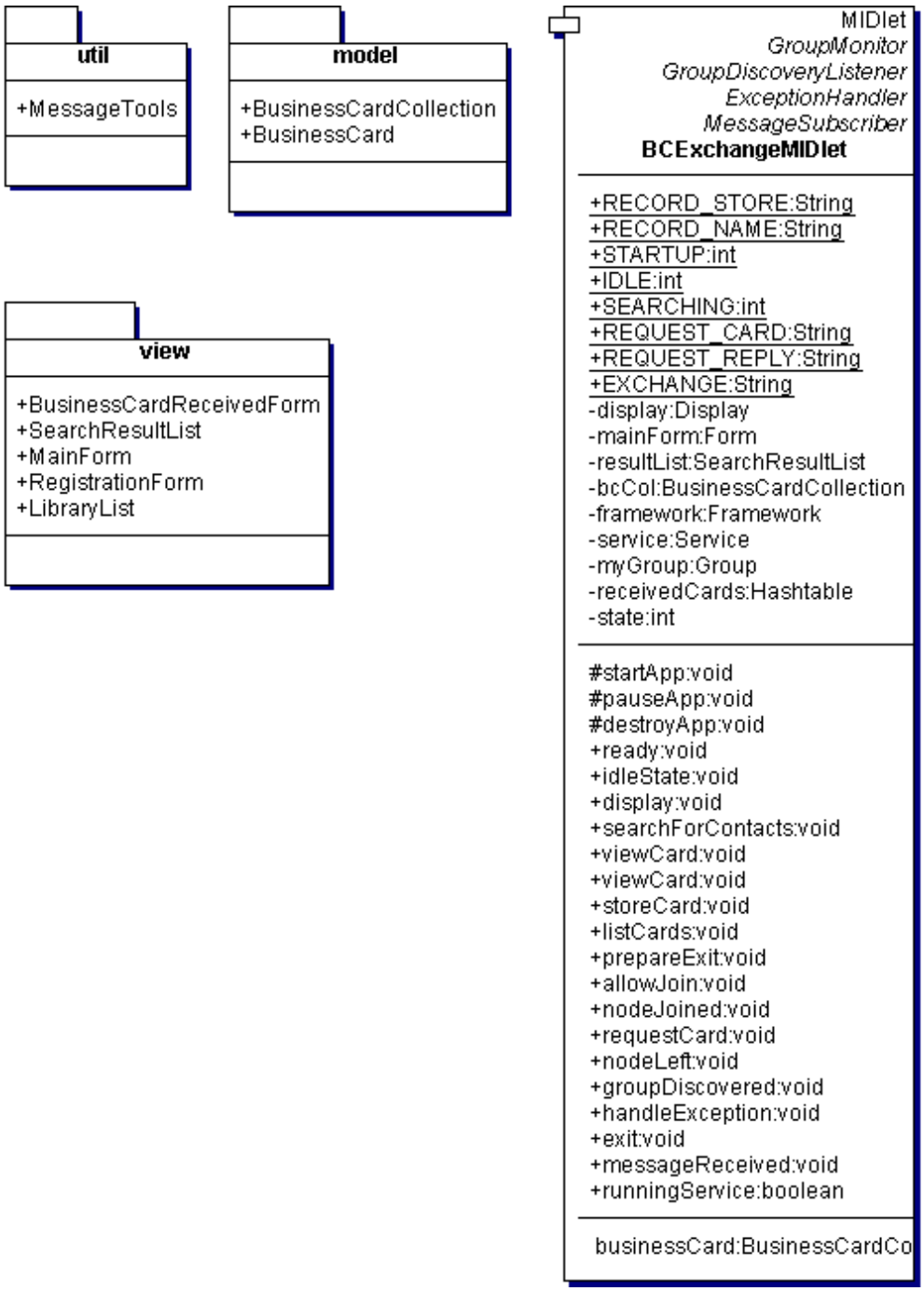


Figure 11.1: UML for the Business Card Exchange MIDlet.

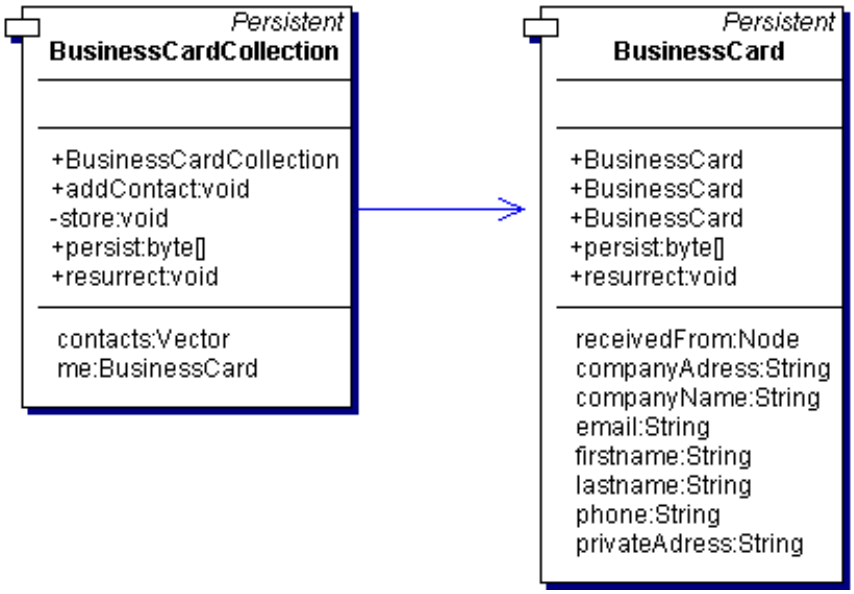


Figure 11.2: UML for the Business Card Exchange MIDlet's model package.

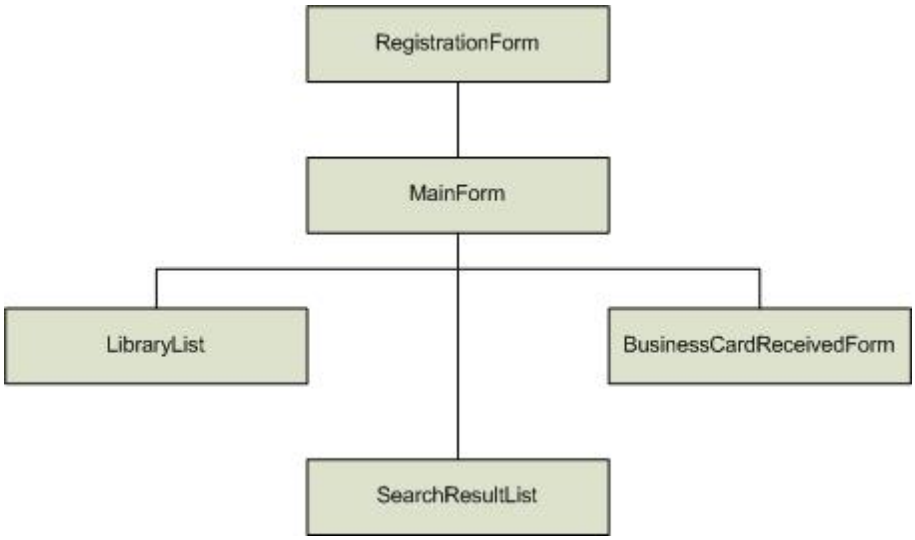


Figure 11.3: Overview of graphical user interface in the Business Card Exchange application.

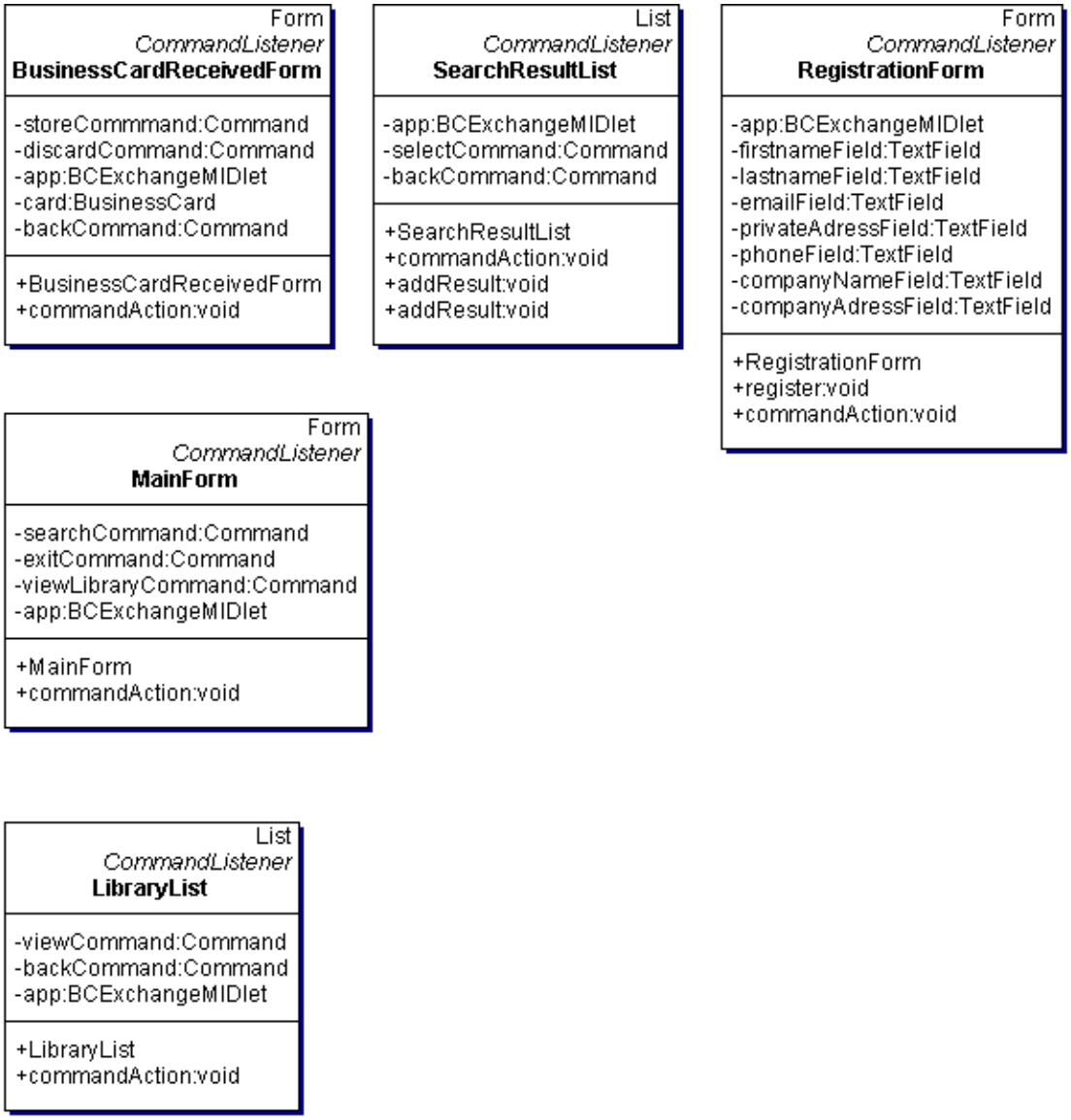


Figure 11.4: UML for the Business Card Exchange MIDlet's view package.

Figure 11.5: UML for the Business Card Exchange MIDlet's util package.

Aspect:	Value:
Lines of code:	325
Number of classes:	9
Number of interfaces:	0
Number of packages:	4
Methods (avg. pr. class)	7.444
Maximum inheritance tree depth:	4

Table 11.1: Business Card Exchange code statistics.

11.3.3 The Util Package

The util package is not part of the ordinary Model View Controller pattern, but contains support functionality for the BCEX MIDlet. The package contains one class, *MessageTools*. The MessageTools class contains methods for translating between *Message* and BusinessCard objects.

11.4 Implementation

This section describes the implementation details of the BCEX MIDlet. Table 11.1 shows the metrics associated with the source code. The complete source code of this application can be found on the attached CD. The BCEX MIDlet covers all the requirements extracted from the scenario and has been tested to run on the mobile phones. We will now give a description of the protocol used for exchanging business cards between nodes.

11.4.1 Business Card Exchange Protocol

When exchanging business cards with the BCEX MIDlet the following protocol is used:

1. The initiating device sends a REQUEST_CARD message to one or more members of its group.
2. The devices receiving the REQUEST_CARD message sends out a CARD_REPLY message with its business card attached.
3. If the user decides to store the received card, a receipt is sent as an EXCAHNGE message, with the user's card attached.

A graphical representation of the protocol is shown in Figure 11.6.

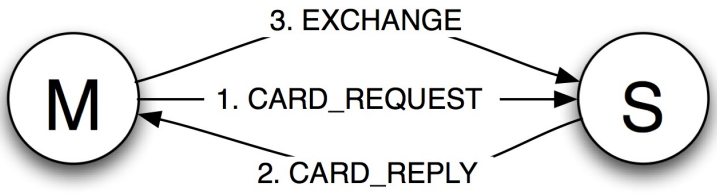


Figure 11.6: The BCEX card exchange protocol.

Chapter 12

PAN Instant Messaging

This scenario was originally included in our previous report [37], focusing on scenario analysis to find requirements for the framework as a whole. In this chapter we will do a scenario analysis where the main focus will be on the user's perspective trying to uncover the requirements for this application only. After elaborating the requirements for the application, an overview of the design will be given followed by a description of the implementation details of the application.

12.1 Scenario

In this scenario, two students attending a lecture at a university want to establish a network between their phones to be able to chat using instant messages.

12.1.1 Goals and Preconditions

The preconditions of this scenario include that the actors each own a mobile phone supporting J2ME applications with MIDP 2.0 and some transfer medium such as ZigBee, Bluetooth or WLAN. Other preconditions for this scenario is to have a group of people, two or more, that are collocated in same place, that want to communicate. The goal of the scenario is to enable these people to communicate directly and instantly with each other by using their mobile phones. Messages are typed in using the same technique as used when writing SMSs and are sent to all the specified participants of a conversation.

The goal of the scenario is to have two or more users connect in a chat group and be able to send instant messages to each other inside this group. This can be stated as:

Goal 1: The application enbables users to create a PAN IM group, to search for and connect to other PAN IM nodes.

Goal 2: The application enables users to communicate directly and instantly with each other inside this group.

Goal 3: The application stores a local profile of the user that is used to identify him or her.

12.1.2 Normal Action Sequence

Peter and Daniel are friends and are both students at a university. They are studying Computer Science and are now attending to a lecture in Compiler Techniques. There are about 150 students almost filling up the auditorium. Peter and Daniel did not arrive at the auditorium at the same time, so now they are sitting two rows away from each other and are not able to communicate by voice. After the professor has introduced the theme of today's lecture, lexical analysis, Peter and Daniel both become quite frustrated. They know all about lexical analysis because they had a project on this in another course. Although they feel, that they can not leave the lecture because the professor is going to talk about the exam the last quarter.

Peter decides to invite Daniel to an ad hoc conversation. He starts his PAN Instant Messaging (PAN IM) application on his mobile phone. Because it is the first time that he uses this application he is asked to enter a nickname and some other personal information that are saved for later use. Then he creates a new chat group and starts a search for devices. The application finds Daniel's device, presents it to Peter and he chooses to establish a connection to this device. Daniel has already started the PAN IM and registered as a user waiting for incoming requests. Now that Peter has established a connection to Daniel, Daniel is alerted, the mobile phone in his pocket vibrates and he accepts the request for a conversation from Peter. Now Peter and Daniel can communicate, and they go on discussing a project in another course for the next half hour. This conversation is of course free of charge because they are using an ad hoc network instead of a wireless cellular network like the GSM network.

During the conversation, a third person named Bill, also starts the PAN IM application. Bill registers as a user and thereby makes himself discoverable for other users using PAN IM. Every now and then Peter searches for new devices nearby that may want to join his and Daniels conversation. Because Bill has registered himself as an PAN IM user, he is discovered by Peter, and Peter are asked if he will allow Bill to join their conversation. Peter feels that the conversation between him and Daniel is kind of private, so he simply rejects Bill from joining them. After a while, when the professor start talking about the exam, Peter and Daniel close the application and starts listening to the professor again.

12.1.3 Critical Exceptions and Error Checking

After a group of users have joined a chat group and are connected to each other, one or more nodes may go down for some reason. A user might just disconnect or a node may go down because of unknown circumstances such as the phones getting to far apart, disturbances in the air or one of the phones going out of power. In these cases all other nodes should be alerted and told that one or more node disconnected.

12.2 Requirements

We will now use the method described in Chapter 2 to analyze the scenario in order to uncover the functional requirements for the application.

12.2.1 Goal Analysis

We will now perform the goal analysis on each of the three goals stated above.

Question 1

The stated goals require computerized support in order to be achieved. This leads to five functional requirements:

FR1: The application must be able to register as a PAN IM node.

FR2: The application must be able create a PAN IM group.

FR3: The application must be able to search for available PAN IM nodes nearby.

FR4: The application must be able to connect to selected PAN IM nodes.

FR5: The application must be able store a user profile locally and permanently.

Question 2

By analyzing the goals we find the following quality or performance requirements:

NFR1: The applications must be able to send and deliever messages in real time.

Question 3

This question does not apply to any of the goals stated in the scenario.

Question 4

None of the goals require any management decisions about resources or responsibilities.

Question 5

Goal 1 and Goal 2 can not be fully automated by the application. These goals have to be triggered by the users of the application. Goal 3 should be done automatically by the application which leads to the following requirement:

FR6: The user profile should automatically be saved and retrieved when needed.

12.2.2 Inbound Event Analysis

In the scenario we can identify the following events:

In 1: Peter enters a nickname and some other personal information that are saved for later use.

In 2: Peter creates a PAN IM group and starts a search for devices.

In 3: Peter chooses to establish a connection to Daniel's device.

In 4: Daniel accepts the request for a conversation from Peter.

Requirements Elaboration

The inbound events result in the following requirements.

FR7: The application must prompt the user for some user profile information the first time it is started. (From In 1)

FR8: The application must receive an answer from the user whether it allows a node to connect or not. (From In 4)

In 2 is covered by FR2 and FR3. In 3 is covered by FR4.

12.2.3 Categorize System Output

The following outputs from the system have been identified:

Out 1: After the search for devices the application presents all available nearby devices to Peter.

Out 2: Daniel is alerted when Peter tries to connect to him and is given the possibility to accept or not accept the connection request.

Requirements Elaboration

These outputs give us the following the functional requirements:

FR9: After searching for available PAN IM nodes, the application must present a list from which the user can choose which PAN IM nodes to connect to. (From Out 1)

FR10: The application must give the user an input request when a incoming connection request is received (From Out 2)

12.2.4 Summary

This gives us the following requirements for the application:

FR1: The application must be able to register as a PAN IM node.

FR2: The application must be able create a PAN IM group.

FR3: The application must be able to search for available PAN IM nodes nearby.

FR4: The application must be able to connect to selected PAN IM nodes.

FR5: The application must be able store a user profile locally and permanently.

FR6: The user profile should automatically be saved and retrieved when needed.

FR7: The application must prompt the user for some user profile information the first time it is started.

FR8: The application must receive an answer from the user whether he or she allows a node to connect or not.

FR9: After searching for available PAN IM nodes, the application must present a list from which the user can choose which PAN IM nodes to connect to.

FR10: The application must give the user an input request when an incoming connection request is received.

NFR1: The application must be able to send and deliver messages in real time.

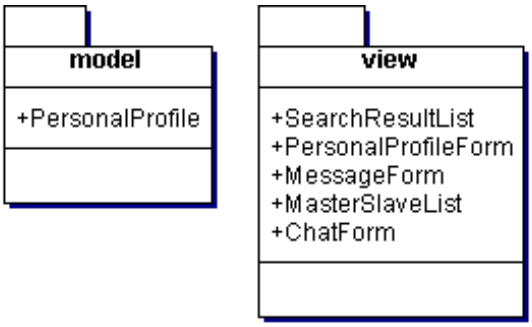
12.3 Design

The controller class of this application is *PanIM*, see Figure 12.1. The PanIM class extends the MIDlet class from MIDP 2.0 and thereby implements the abstract methods from this class. It also implements a lot of the Peer2Me interfaces and holds all functionality that utilizes the framework.

When a user starts this application, he or she will have to choose between the role of being slave or master. This choice will control what kind of functionality that will be available for that user while running the application with the chosen role.

12.3.1 The Model Package

The only class in the Model package is the *PersonalProfile* class, shown in Figure 12.2. All users of the PanIm application has a personal profile. This is stored on the user's mobile phones. The personal profile holds information about a user like his or her nickname, real name and e-mail address. The class is a typical JavaBean with getter and setter methods and holds functionality for making itself persistent by using the Persistence Layer in Peer2Me.



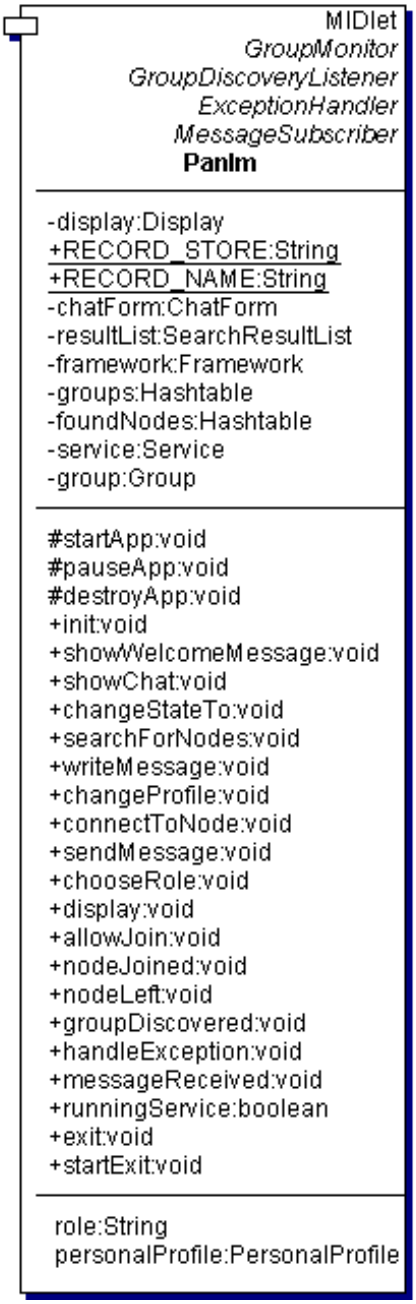


Figure 12.1: Overview of the classes and packages of PAN IM.

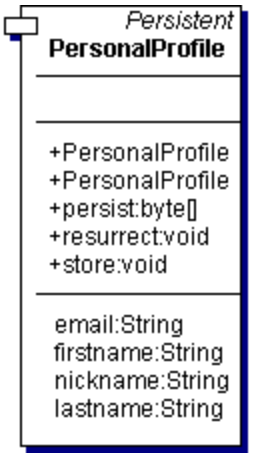


Figure 12.2: Overview of the classes in the model package.

12.3.2 The View Package

The view package, shown in Figure 12.4, holds all the classes that make out the graphical user interface of the application. As mentioned, the application is controlled by a controller class that works as a state machine. For each state there is a corresponding user interface class. The user interface hierarchy is shown in Figure 12.3. Each of these classes will now be described individually:

ChatForm: This is the main view of the application and also the starting point. This view shows personal messages from other users and control messages from the system. From this view you can also choose to start writing a new message, start a search after new nodes or edit your personal profile.

MasterSlaveList: This is the first view that is presented for the user when starting the application. In this view the user can choose between the roles of master and slave.

MessageForm: In this view it is possible to construct a message and start the process of sending it to the other nodes.

PersonalProfileForm: This view shows the user's personal profile. The data in this form can also be edited and stored.

SearchResultList: When searching for new nodes to join the chat group, the framework will return possible nodes nearby to join one by one. These nodes will be presented in this view. It will further be possible to select which nodes that should be allowed to join the group.

12.4 Implementation

This section describes the implementation details of the PAN Instant Messenger application. Table 12.1 shows the metrics associated with the source code. The complete source code of this applica-

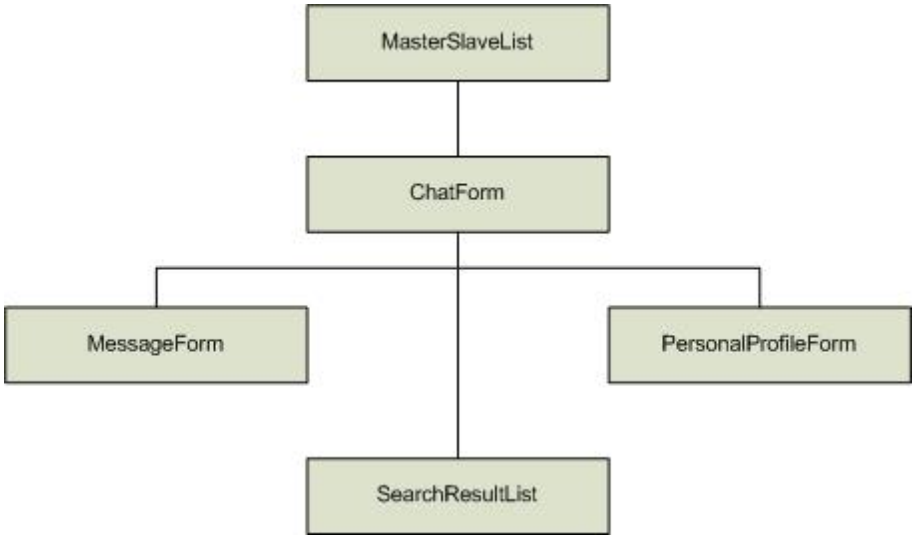


Figure 12.3: Overview of gui in the PAN Instant Messaging application.

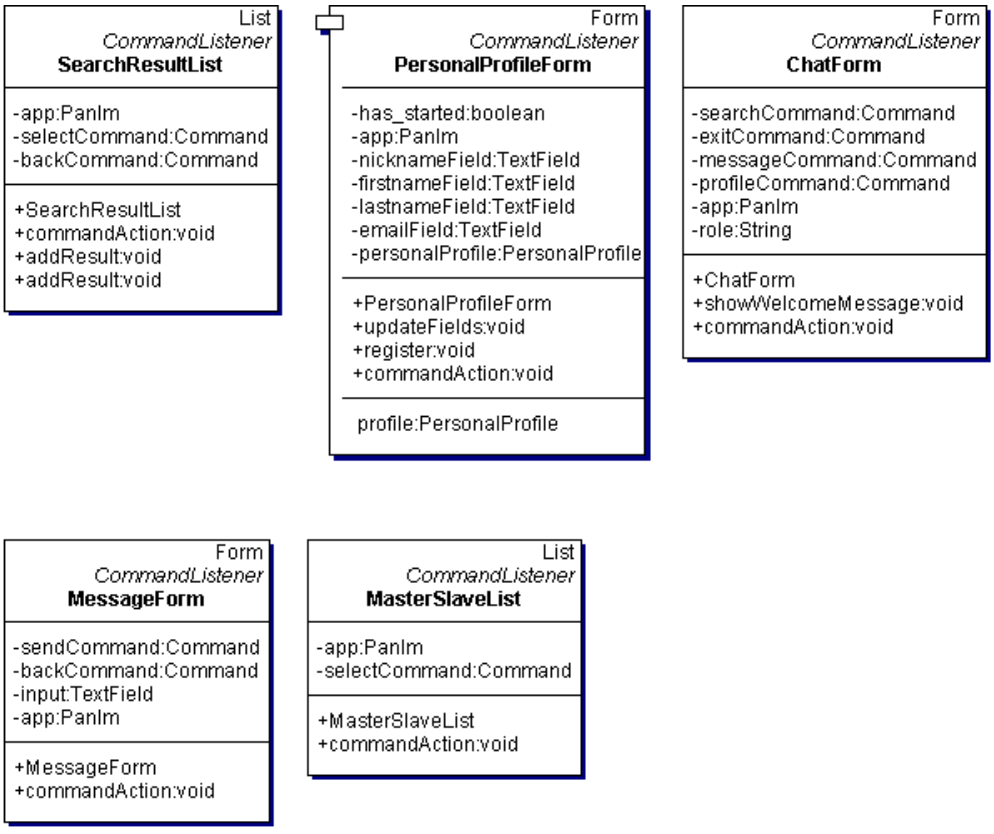


Figure 12.4: Overview of the classes in the view package.

Aspect:	Value:
Lines of code:	242
Number of classes:	7
Number of interfaces:	0
Number of packages:	2
Methods (avg. pr. class	8
Maximum inheritance tree depth:	4

Table 12.1: PAN IM code statistics.

tion can be found on the attached CD. The application covers all the requirements extracted from the scenario, except FR8 and FR10. We have chosen to not implement these two requirements because we wanted to make the process of creating a chat group more automatic and faster, ignoring input requests from the slave chat nodes in the joining process. The application has been tested to run perfectly on the mobile phones.

When implementing this application, we realized that it really does not need a lot of special application logic. PAN IM is a brilliant example of an application that utilizes the functionality of Peer2Me one hundred percent. A major part of the functional requirements are covered by the framework functionality. The most important concerns when implementing this application was to create some well functioning user interfaces and to make a simple infrastructure for holding information regarding a chat group. One particular algorithm that we would like to mention had to be deployed for obtaining the right functionality. This algorithm for filtering out already connected nodes is described below.

12.4.1 Filtering Already Connected Nodes

When a master node's user searches for available PAN IM nodes nearby he will receive a list of available nodes. It is important that the nodes that are already connected to this device are not shown in this list. This is done by performing a filtering of devices when receiving join messages. The code responsibility for this filtering is important for the application to function properly and is also a good example of how to compare already connected nodes with nodes trying to join. The code, shown in Listing 12.1, is situated in the allowJoin method in the PanIm class. This method is called by the framework when a new node tries to join the group. The parameters of the method hold the node that tries to join and the group that the node tries to join. The filtering is performed and the node is put in the search result list if it is not present in the group already.

Listing 12.1: The allowJoin method that contains code to filter out already connected nodes.

```
1  boolean found = false;
2  Vector slaves =  service.getGroup().getSlaves();
3
4  for(int i=0;i<slaves.size();i++){
5      Node existingNode = (Node)slaves.elementAt(i);
6      if(existingNode.getNodename().equals(node.getNodename())){
7          found = true;
8      }
9  }
10
```

```
11 if (!found){
12     foundNodes.put(node.getNodename(),node);
13     resultList.append(node.getNodename(),null);
14 }
```

Chapter 13

Converging top ten list

This scenario was originally included in our previous report [37], focusing on goal analysis to find requirements for the framework as a whole. In this chapter we will do a goal analysis to where the main focus will be on the user's perspective trying to uncover the requirements for this application only. After elaborating the requirements for the application, an overview of the design will be given followed by a description of the implementation details of the application.

13.1 Scenario

In today's world with ever toughening competition in various markets, the consumers benefit from knowing which service provider or product is the best. The converging top ten list application enables people to collaborate in automatically finding the best services or products available in the market. This scenario describes how users in a population can register information in a top ten list form on their mobile devices. When they meet with other users from the same population, their lists are automatically exchanged and compared causing each user's list to converge towards the true top ten list. As an example here, to extract requirements for the application, we will use a scenario where students try to find out which bar has the lowest beer prices. The students automatically exchange top ten lists of beer prices when walking around the campus area.

13.1.1 Goals and Preconditions

The preconditions of this scenario include the mobile phones that the students carry around with them as they move around on the campus area. The mobile phones have to be able to make use of some kind of transfer medium such as Zigbee, WLAN or Bluetooth and have support for J2ME and MIDP 2.0.

Each student will have a top ten list stored on their mobile phone. The goal is to have all the various lists converged into one.

The goals for this scenario are the following:

Goal 1: The application should let a student register, store and maintain various kinds of top ten list.

Goal 2: The application should automatically connect to other Converging Top Ten List applications and exchange relevant top ten lists.

Goal 3: The application should use the received lists and the old lists as input to calculate new top ten lists and store them on the phone.

13.1.2 Normal Action Sequence

A student maintains a local list of the best beer prices known to him at the moment on his phone. Whenever the student visits a bar with beer prices that changes this list, he registers the price as well as the date this price was valid. Obviously, one student is unable to visit every available bar. This is where collaboration with others come in.

When the students move around on the campus area, they enable their phones to be available for information exchange with other students running the same application.

As two students running the Converging Top Ten List application pass in the hallway or sit next to each other during a lecture, their mobile phones will sense that they are in proximity and providing the same service. Without the need of any interaction from the students, the applications will connect and exchange their top ten lists. On each of the phones, these two lists will be compared to produce a new list with the ten best prices from the two original lists. Information about the transactions and the calculations will be given in the application user interface. As time goes by and more and more students exchange lists by chance encounters, everyone's lists will converge into the same top ten list giving everyone up to date information on where to find the best beer prices. Lists will be identified by a special category identifier. Only lists marked with the same category will be compared and merged.

Critical Exceptions and Error Checking

If an encounter between to students ends to soon, chances are that the discovery and transfer will not have time to finish. In the case where the encounter is ended before a connection is opened, no measures will have to be taken. In the case where the transmission of the top ten list is aborted due to errors, interference or other problems, two different strategies can be used:

- The entire list can be discarded as if the encounter never had taken place.
- If, for instance, the first three list entries have been transferred before communication fails, these three entries can be merged with the local list, which leads to the top three entries moving closer to the converged list.

13.2 Requirements

We will now use the method described in Chapter 2 to analyze the scenario in order to uncover the functional requirements for the application.

13.2.1 Goal Analysis

We will now perform the goal analysis on each of the three goals stated above.

Question 1

The stated goals require computerized support in order to be achieved. This leads to five functional requirements:

FR1: The application must enable users to create new top ten list within a specified category.

FR2: The application must enable users to store and maintain top ten lists.

FR3: The application must be able to search for other Converging Top Ten applications nearby automatically.

FR4: The application must be able to exchange top ten lists with other applications.

FR5: The application must be able to calculate a new top ten list from two old lists.

FR6: The application must be able to automatically store the new calculated lists permanently.

Question 2

The goals do not describe any performance or quality properties.

Question 3

This question does not apply to any of the goals stated in the scenario.

Question 4

None of the goals require any management decisions about resources or responsibilities.

Question 5

Goal 2 and goal 3 should be fully automated by the application. This is stated in the above requirements. Goal 1 and its associated tasks must be triggered by a user.

13.2.2 Inbound Event Analysis

In the scenario we can identify the following events:

In 1: A student registers the price and the the date this price was valid.

Requirements Elaboration

The inbound events result in no new requirements. In-1 is covered by FR1 and FR2.

13.2.3 Categorize System Output

The following outputs from the system have been identified:

Out 1: Information about the transactions and the calculations will be given in the application user interface.

Requirements Elaboration

This output leads to a new functional requirement:

FR7: When transactions or other events occur, the application must present information about them to the user.

13.2.4 Summary

This gives us the following requirements for the application:

FR1: The application must enable users to create new top ten list within a specified category.

FR2: The application must enable users to store and maintain top ten lists.

FR3: The application must be able to search for other Converging Top Ten applications nearby automatically.

FR4: The application must be able to exchange top ten lists with other applications.

FR5: The application must be able to calculate a new top ten list from two old lists.

FR6: The application must be able to automatically store the new calculated lists permanently.

FR7: When transactions or other events occur, the application must present information about them to the user.

13.3 Design

The design of this application is centered around the controller class, TopTenListApp (see Figure 13.1). This class constitutes the actual MIDlet by subclassing the MIDlet class included in MIDP 2.0. This class also implement different interfaces from Peer2Me and thereby implements different methods required by Peer2Me. It holds methods to control the program flow and logic. These

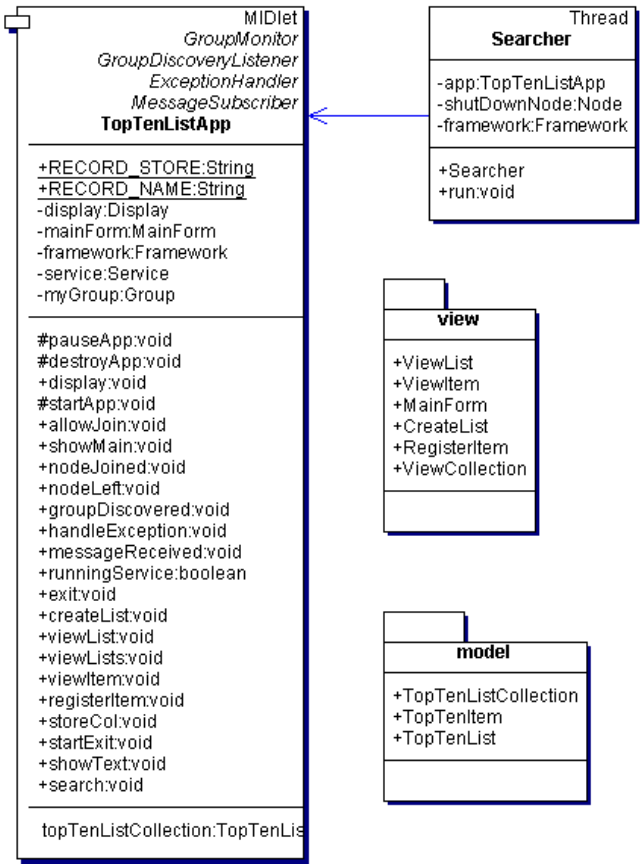


Figure 13.1: Overview of the classes and packages of the Converging Top Ten List applications.

methods are called by the different views to make the program switch between different states. The Top Ten List application differs from the two other example applications because it is based upon the concept of automatic exchange of information. This mechanism is also controlled by the TopTenListApp class. The class has the responsibility for automatically starting a new search for devices when the last one has finished. To obtain this functionality, the class gets help from another class called *Searcher*.

13.3.1 The Model Package

The model package holds the classes that represent the data model of this application, see Figure 13.2. A user of this application can create and maintain a number of different top ten lists. A top ten list is described by its identifier and description text. All these top ten lists are stored in a top ten list collection. Each top ten list is comprised of top ten items. A top ten item holds information about the name of the product or service, the date of the registration and the price that where registered. These three entities are implemented as three classes called *TopTenItem*, *TopTenList* and *TopTenListCollection*.

The TopTenItem class is a typical Java Bean that holds some information and provides simple

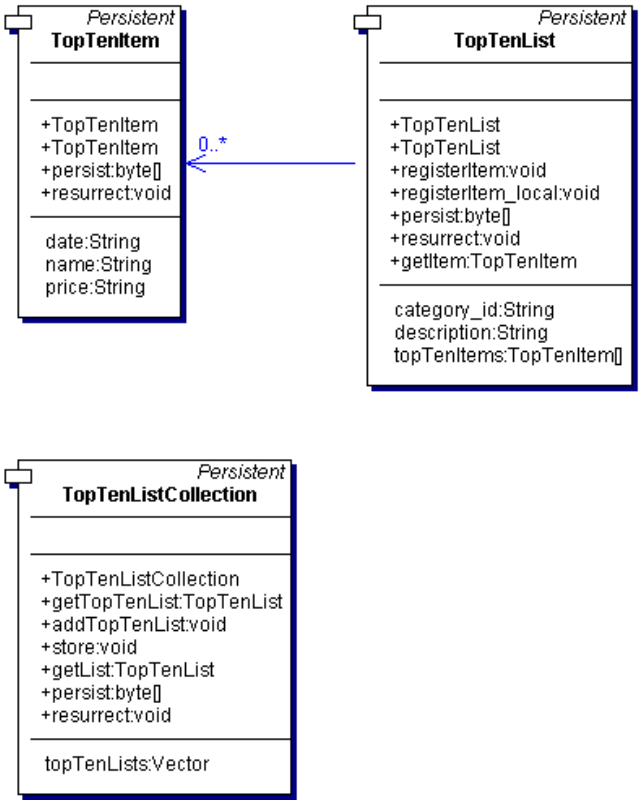


Figure 13.2: Overview of the classes in the model package.

getter and setter methods. The TopTenList, in addition to containing an array of TopTenItems, also holds some logic. The *registerItem* method has the responsibility for checking if a new item has values that makes it a top ten item, and if that is the case, the top ten list will be reorganized and the last item will be thrown out of the list.

All the classes are persistent and therefore implements the methods defined in the *Persistent* interface of the Persistence Layer of Peer2Me.

13.3.2 The View Package

The view package holds all the classes that make out the user interfaces or information screens in the application. As mentioned, the whole application is controlled by a controller class that works as a state machine. For each state there is a corresponding user interface class. The user interface hierarchy is shown in Figure 13.3. A UML diagram showing the classes in the package is shown in Figure 13.4. Each of these classes will now be described individually:

MainForm: This is the view that is shown when the application is running normally. Different kinds of information is displayed through the form, for instance information about received lists and connected nodes.

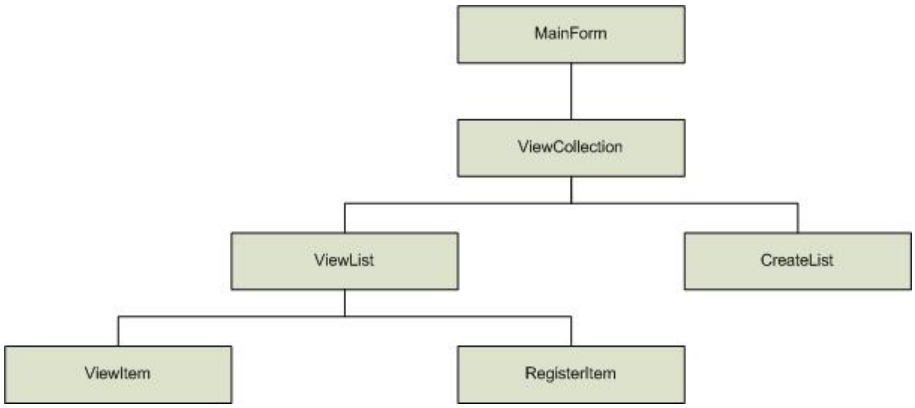


Figure 13.3: Overview of gui in the Top Ten List application.

ViewCollection: From the MainForm, it is possible to view the local collection of top ten lists. This view shows a list of all available top ten lists as selectable items. It is also possible to choose to create a new list from this view.

CreateList: This is a view that enables the user to enter information about a new list and then create it.

ViewList: This view shows a list containing all the items that the list holds. Each item is selectable.

ViewItem: When an item is selected in the ViewList view, this view is shown. The view displays all the attributes of a top ten item, but it is not possible to edit this information.

RegisterItem: This is a view that enables the user to enter information about a new top ten item and then register it in the top ten list.

13.4 Implementation

This section describes the implementation details of the Converging Top Ten List application. Table 13.1 shows the metrics associated with the source code. The complete source code of this application can be found on the attached CD. The application covers all the requirements extracted from the scenario and has been tested to run on the mobile phones. It should be remarked that the requirement FR4 should be extended before more full scale testing of this applications. In its current state the application just exchanges one list. The application should be extended with functionality for the user to select which lists on his phone that should be automatically exchanged. We will now explain some of the most important implementation details of this application.

13.4.1 The Sorting Algorithm

The registerItem method in the TopTenList class is the heart of the sorting functionality of the application. This is used when the user updates or creates a new item in a list. The method

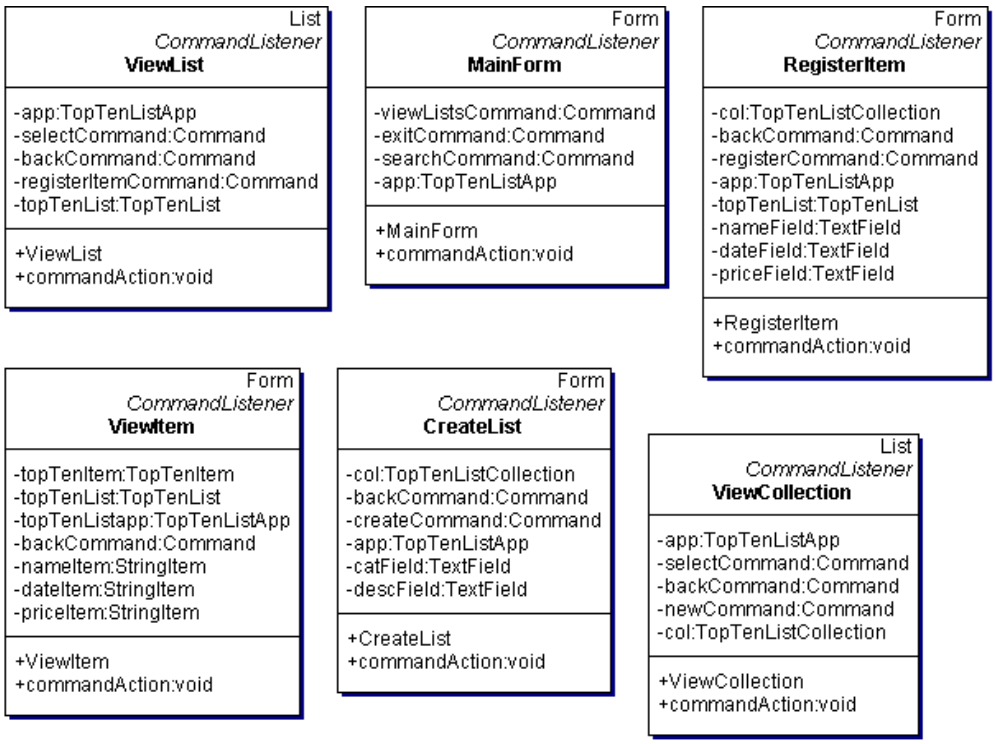


Figure 13.4: Overview of the classes in the view package.

Aspect:	Value:
Lines of code:	454
Number of classes:	11
Number of interfaces:	0
Number of packages:	2
Methods (avg. pr. class	5.363
Maximum inheritance tree depth:	4

Table 13.1: Top Ten List statistics.

ensures that the list will remain sorted after the edit. The algorithm embedded in the method uses a method similar to bubble sort for sorting the elements in the list, shown in Listing 13.1. When the application receives a list from another phone it has to compare this list with the one already stored on the phone. Item by item is picked out from the received list and is then registered, by using the registerItem method, in the other list. By doing this for all the top ten items in the received list, the new list will be updated with the best prices from the both the original lists.

Listing 13.1: The registerItem method that puts an item on its right place in a list.

```
1  public void registerItem(TopTenItem topTenItem){
2
3      ...
4
5      for(int i=0;i<topTenItems.length;i++){
6          int priceOne = Integer.parseInt(topTenItem.getPrice());
7          int priceTwo = Integer.parseInt(topTenItems[i].getPrice());
8          if(priceOne<priceTwo){
9              topTenItems[i] = topTenItem;
10             pos = i;
11             cheaper=true;
12             break;
13         }
14     }
15
16     if(cheaper){
17         if(pos<(topTenItems.length - 1){
18             for(int i=(pos+1);i<topTenItems.length;i++){
19                 topTenItems[i]=back_items[i-1];
20             }
21         }
22     }
23 }
```

13.4.2 Proactive Disconnection Message

The different phone models and Bluetooth API implementations tend to vary a lot in how they function. For instance, some phones detect disconnections of other phones very quickly (a few seconds) and others spend 15-20 seconds before detecting a disconnection. Because a transaction (exchanging the lists) with the Converging Top Ten List application has to be performed very quickly, we designed a special disconnection mechanism to ensure that the transaction would finish a soon as possible.

This is done by sending out an extra message after receiving a top ten list. When the other phone receives this disconnection message, it knows that it can disconnect this node right away. In this way we speed up the disconnection and thereby also speed up the whole transaction by an order of magnitude.

13.4.3 Automatic Searching

In the design part we described a Searcher class that holds the functionality that does the automatic searching for other nodes. This was solved by using a simple java thread. The Searcher class

extends the Thread class from CLDC 1.0 API. Every time a search after nodes is finished an instance of this class is created to start a new search.

Part IV

Developing Peer2Me Applications

Chapter 14

Peer2Me Development Guide

This chapter assumes the reader is familiar with the central concepts of writing MIDlets for the J2ME platform. It provides a thorough walkthrough on how to write ad hoc networked MIDlets using the Peer2Me framework. The entire chapter is aimed at developers that want to learn how to use Peer2Me and is therefore written in a less formal way than the rest of the report. This process will be explained by an example describing two MIDlets, *SlaveMIDlet* and *MasterMIDlet*. We will not explain anything about the GUI code used in the code examples. The full source code for each of the applications can be found in Appendix C.

14.1 Central Concepts

In the Peer2Me framework there are a couple of concepts that need to be understood before you can write a Peer2Me MIDlet. We will now give an overview of these concepts.

Service Central to Peer2Me is the service. Devices that communicate must all be running the same service. In the Peer2Me implementation, services are represented by the class *Service* in the *no.ntnu.idi.mowahs.project.domain* package. A service normally represents a specific application type, for instance a chat application. When the framework searches for other nodes, it searches for other nodes running the same service (which means that they are running the same application).

Node A device running a service is called a node. Nodes are represented by the class *Node* in the *no.ntnu.idi.mowahs.project.domain* package.

Group When nodes are connected in a network they form a group. A group has a name and a description and is represented by the class *Group* in the *no.ntnu.idi.mowahs.project.domain* package. A group is associated with a service.

Message Nodes connected in a group can exchange messages. Messages can contain any number of message parts, each with a description and a value. Messages and message parts are represented by the classes *Message* and all subclasses of *MessagePart* in the *no.ntnu.idi.mowahs.project.domain* package respectively.

Framework The core in Peer2Me is the actual framework. The MIDlet interacts with other nodes on the network through the framework. The use of the framework abstracts all network functionality from the MIDlet. The framework is represented by the class *Framework* in the *no.ntnu.idi.mowahs.project.domain* package.

14.2 Starting the MIDlet

All MIDlets written for J2ME are subclasses of the more general *MIDlet* class provided in the *javax.microedition.midlet* package. MIDlets are executed as a simple state machine with three methods for changing the state of the MIDlet. These three methods are:

protected void startApp() throws MIDletStateChangeException This method is called when the MIDlet is first started, it is used to initialize the MIDlet's context.

protected void pauseApp() throws MIDletStateChangeException If the MIDlet is temporary set on hold by an incoming call, sms, etc it is paused through this method.

protected void destroyApp() throws MIDletStateChangeException When the MIDlet exits, its context is cleaned up by this method.

In this tutorial we will focus on using the *startApp* and *destroyApp* methods. First we will take a look at what happens when the MIDlet is started.

14.2.1 Initializing the Framework

Before anything can be done with the Peer2Me framework, it must be initialized. Peer2Me needs certain properties to be set, such as node name, node description and preferred network. This is achieved through the code shown in Listing 14.1, where *myName*, *myDescription* and *networkName* are String objects giving the local node name, the node description and the Java name of the network module to be loaded respectively.

Listing 14.1: Initializing the Framework.

```
1    myFramework = Framework.getInstance(myName, myDescription, networkName);
2    myFramework.init();
```

In Listing 14.1, the *Framework* instance is first retrieved by feeding the node name, the node description and the network name to the *getInstance* method. It is the initialized by the *init* method. The init method prepares the framework for use with the arguments given previously. *NetworkName* is a string containing the Java name of the network module to load. For more information about loading a network module take a look at Section 14.8 later in this chapter.

14.3 Slave vs. Master

After Peer2Me has been initialized it is ready for use. A device in a Peer2Me network can run either as a slave or as a master. Whether a device runs as a slave or a master has significance when it comes to how it handles groups. Master devices are group administrators handling connections to slave nodes, deciding who should be allowed to join the group and routing messages among the slaves in a group. Lets start with the Slave case.

14.3.1 Slave

After initializing its Peer2Me instance, a slave device must register a service which it will be listening for connections on. This can be done by including the code line shown in Listing 14.2. In Listing 14.2, *myService* is a *Service* object containg the name of the service.

Listing 14.2: Registering a service with Peer2Me.

```
1 myFramework.registerService(myService);
```

14.3.2 Master

The master node in a Peer2Me network is a bit more complex functionality wise than the slave node. The master has to register a service with the framework the same way as the slave node does, shown in line 1 of Listing 14.3. In addition, a master has to be associated with a group before doing anything else. To create a group, simply instanciate an object of the Group class. Then set yourself as the master of the group, register yourself as group monitor and finally relate the group to a service and vice versa. These steps are shown in Listing 14.3 on line 2-6. The master is then free to search for available slaves in its proximity. The code line for searching for other nodes running the same service in slave mode is done through the last line of Listing 14.3.

Listing 14.3: Searching for other nodes running as slaves.

```
1 myFramework.registerService(myService);
2 myGroup = new Group();
3 myGroup.setMaster(myFramework.getLocalNode());
4 myGroup.setMonitor(this);
5 myService.setGroup(myGroup);
6 myGroup.setService(myService);
7 myFramework.startGroupSearch(myService);
```

14.4 Discovering Groups

When a listening slave is discovered by a master, the slave is notified of the available group. This is done through the method *public void groupDiscovered(Group groupFound)* method defined by the *GroupDiscoveryListener* interface. All MIDlets that are to run as slaves must therefore implement this interface and register itself with the framework instance through its *setGroupDiscoveryListener* method.

In our case, we want the slave device to attempt to join a group as soon as it is found. Joining groups is achieved through the *joinGroup(Group)* method in the *Framework* class. The complete *groupDiscovered(Group)* method is shown in Listing 14.4. The first line in the method, sets the group's monitor (we'll get back to this later). The next few lines just prints the members of the group to screen before the last line asks to join the group.

Listing 14.4: The groupDiscovered method in the TestSlave class, including GUI code.

```
1   public void groupDiscovered(Group group) {
2       group.setMonitor(this);
3
4       writeText("Received␣group!");
5       writeText("Master:" + group.getMaster().getKey());
6       Enumeration enum = group.getSlaves().elements();
7       while(enum.hasMoreElements()){
8           writeText("Slave:␣" + ((Node)enum.nextElement()).getKey());
9       }
10
11      myFramework.joinGroup(group);
12  }
```

The master node is the one who controls the group in the first place, which means that the group is known to the master device at all times. This means that the master does not need to implement the *GroupDiscoveryListener* interface.

14.5 Handling Dynamic Groups

An application running over an ad hoc network is prone to have a dynamic degree of connectivity. The *Group* object is a representation of all the nodes the current node can reach trough its master. If remote nodes disconnect or go out of reach or if a new node joins the group, all the nodes in the group must be notified of the event. In Peer2Me, the handling of dynamic groups is managed at the framework level, but the application must be notified of such events. In order to do this, Peer2Me provides an interface for notifying applications about changes in groups, the *GroupMonitor* interface. GroupMonitor provides three methods, *nodeJoined(Group, Node)*, *nodeLeft(Group, Node)* and *allowJoin(Group, Node)*. The first two are just in the case of a node joining the group and a node leaving the group respectively. The third is only used on the master node in the case of a slave wanting to join a closed group.

On the slave device the joining of a new node will trigger a message to the new node so we will save this for later.

In this code example, in the case of the master device, both the joining and leaving of nodes are only written to the screen, which gives us the method implementations shown in Listing 14.5.

Listing 14.5: Handling group changes in TestMaster.

```
1
2   public void nodeJoined(Group group, Node node) {
3       writeText("Node␣joined:␣" + node.getKey());
4
5   }
```

```
6
7
8       public void nodeLeft(Group group, Node node) {
9           // TODO Auto-generated method stub
10          writeText("Node left: " + node.getKey());
11
12      }
```

14.6 Sending and Receiving Messages

In Peer2Me, messages are represented as *Message* objects. A message has a sender, a set of recipients and multiple message parts. In Listing 14.6 a message is created and sent when a new node joins the group. First a message is displayed on screen, stating that a new node has joined the group. Over the next lines, a message is created, filled with one message part and the new node set as recipient. The last line of the method sends the message by handing it over to the *sendMessage(Message, Service)* method.

Listing 14.6: Join notifications in TestSlave.

```
1       public void nodeJoined(Group group, Node node) {
2           writeText("Node joined: " + node.getKey());
3
4           Message message = new Message();
5           TextMessagePart text = new TextMessagePart();
6           text.setDescription("message");
7           text.setFieldValue("hello slave");
8           message.addMessageBodyPart(text);
9           message.addRecipient(node);
10
11          myFramework.sendMessage(message, myService);
12      }
```

Peer2Me provides an interface called *MessageSubscriber*. When the Peer2Me framework receives an application message from a remote node, the object registered as a MessageSubscriber will be notified through its *messageReceived(Message)* method.

14.7 Handling Exceptions

Peer2Me is multithreaded in addition to being an easy victim to network failures, due to the nature of ad hoc networks. When something goes wrong, there is a way for the MIDlet to subscribe to the error messages produced and letting it decide which errors to show to the user. The interface ExceptionHandler in the *no.ntnu.idi.mowahs.project.framework* package defines one method, *handleException(Exception)*. By implementing this interface, the MIDlet can register as a ExceptionHandler with the *setExceptionHandler(ExceptionHandler)* method in the *Framework* class.

In both *TestMaster* and *TestSlave*, all we will do with Exceptions are to print the error message to the display. Listing 14.7 shows the *handleException(Exception)* method.

Listing 14.7: Exception handling.

```
1     public void handleException(Exception e) {
2         writeText(e.getMessage());
3
4     }
```

14.8 Choosing the Right Network Module

When initiating Peer2Me, the name of the preferred network module must be provided to the *getInstance* method in the *Framework* class. Peer2Me's fundamental architecture is highly modular supporting the introduction of new network modules without actually changing the Peer2Me source code. Because of this modularity, Peer2Me must be told which network module to load at runtime. Currently the only network module available is the Bluetooth module. The Java-name and the class path of this module is: *"no.ntnu.idi.mowahs.project.bluetooth.network.BluetoothNetwork"*.

14.9 Complete *startApp* Methods

This gives us the full code needed in the slave's *startApp* method, shown in Listing 14.8.

Listing 14.8: The startApp method from the TestSlave class, including GUI code.

```
1     protected void startApp() throws MIDletStateChangeException {
2         display = Display.getDisplay(this);
3         form = new Form("TestSlave");
4         myService = new Service(SERVICE_ID);
5         writeText("Startet testSlave...");
6         myFramework = Framework.getInstance("slavetester", "testing slave", "no
              .ntnu.idi.mowahs.project.bluetooth.network.BluetoothNetwork");
7         myFramework.init();
8         myFramework.registerSlave(myService);
9         myFramework.setMessageSubscriber(this);
10        myFramework.setExceptionHandler(this);
11        myFramework.setGroupDiscoveryListener(this);
12
13    }
```

This gives us the full code needed in the master's *startApp* method, shown in Listing 14.9.

Listing 14.9: The startApp method in the TestMaster class, including GUI code.

```
1     protected void startApp() throws MIDletStateChangeException {
2         display = Display.getDisplay(this);
3         form = new Form("TestMaster");
4         myService = new Service(SERVICE_ID);
5         myGroup = new Group();
6
7         writeText("Startet testMaster...");
8         myFramework = Framework.getInstance("mastertester", "testing master", "
              no.ntnu.idi.mowahs.project.bluetooth.network.BluetoothNetwork");
9         myFramework.init();
```

```
10            myFramework.setMessageSubscriber(this);
11            myFramework.setExceptionHandler(this);
12
13            myGroup.setMaster(myFramework.getLocalNode());
14            myGroup.setMonitor(this);
15            myService.setGroup(myGroup);
16            myGroup.setService(myService);
17            myFramework.registerSlave(myService);
18
19            myFramework.startGroupSearch(myService, null);
20
21        }
```

Chapter 15

Using the Persistence Layer in Peer2Me

As described in Chapter 7 and Chapter 8, Peer2Me contains a simple key-oriented persistence layer. In this chapter we will explain how to use the persistence layer when writing applications. This chapter is aimed at application developers and assumes you are familiar with Java programming and know the main concepts associated with J2ME programming.

15.1 Making an Object Persistent

If your application needs to store an object or a set of objects, these objects need to implement the *Persistent* interface provided in the package *no.ntnu.idi.mowahs.project.util*. We will explain this interface by using examples from the implementation of the Business Card Exchange (BCEX) application. BCEX contains two persistent objects, shown in Figure 15.1. We will start with the class *BusinessCard*.

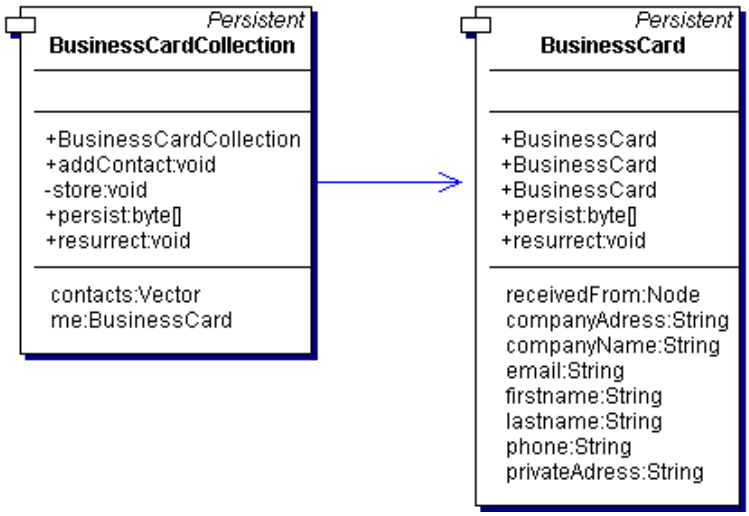


Figure 15.1: The persistent objects of Business Card Exchange.

BusinessCard is created with a standard Java Bean layout. BusinessCard objects have several instance properties accessible through getter and setter methods. In J2ME an object can not be serialized and recreated through the use of reflection such as in J2SE so we have two write our own way of serializing an object. Object serialization is provided through the method *byte[] persist()*. This method translates an object to a byte array in a format itself understands and then returns this byte array. This is the persistent version of the object that can be saved in the applications *RecordStore* through the class PersistenceManager (explained later) associated with an identifying key.

An example showing how to implement the persist method is shown in Listing 15.1.

Listing 15.1: The persist method in BusinessCard.

```
1   public byte[] persist() throws IOException {
2       ByteArrayOutputStream bout = new ByteArrayOutputStream();
3       DataOutputStream dout = new DataOutputStream(bout);
4
5       dout.writeUTF(this.getClass().getName());
6       dout.writeUTF(firstname);
7       dout.writeUTF(lastname);
8       dout.writeUTF(email);
9       dout.writeUTF(phone);
10      dout.writeUTF(privateAdress);
11      dout.writeUTF(companyName);
12      dout.writeUTF(companyAdress);
13
14      dout.flush();
15      return bout.toByteArray();
16  }
```

The Persistent interface also defines a method, *void resurrect(byte[] data)*. Objects that are persistent need an empty constructor method that initializes the object but does not set any of the property values. After creating a new, empty object the persistent byte array is delivered to it through the resurrect method and parsed in order to set the objects properties to the correct values. The format in the supplied byte array will be the same as in the byte array delivered by the same object's persist method, therefore the resurrect method's actions should read data from the byte array in the same order as it was written in the persist method.

An example showing how to implement the resurrect method is shown in Listing 15.2.

Listing 15.2: The resurrect method of BusinessCard.

```
1   public void resurrect(byte[] persistent) throws IOException,
        PersistenceException {
2       ByteArrayInputStream bin = new ByteArrayInputStream(persistent);
3       DataInputStream din = new DataInputStream(bin);
4
5       if(!din.readUTF().equals(this.getClass().getName())){
6           throw new PersistenceException("Error ressurecting class!");
7       }
8
9       firstname = din.readUTF();
10      lastname = din.readUTF();
11      email = din.readUTF();
12      phone = din.readUTF();
```

```
13          privateAdress = din.readUTF();
14          companyName = din.readUTF();
15          companyAdress = din.readUTF();
16
17      }
```

15.1.1 Persistence for Nested Objects

Implementing the *Persistent* interface for a simple object is not a too complex task, but nested objects are not necessarily any harder. In the BCEX application, the second persistent object is *BusinessCardCollection* which is comprised of several *BusinessCard* objects. The solution to this is just to include the nested objects when creating and parsing the byte array.

Listing 15.3 shows how *BusinessCardCollection* objects are made persistent. A BusniessCard-Collection object contains a reference to a single BusinessCard and a vector containing several BusinessCards. When storing one of the business cards, the byte array from its *persist* method is included in the BusinessCardCollection byte array.

Listing 15.3: The resurrect and persist methods of BusinessCardCollection.

```
1       public byte[] persist() throws IOException{
2           ByteArrayOutputStream bout = new ByteArrayOutputStream();
3           DataOutputStream dout = new DataOutputStream(bout);
4
5           dout.writeUTF(this.getClass().getName());
6           dout.writeUTF("me");
7           byte[] data = me.persist();
8           dout.writeInt(data.length);
9           if(data.length > 0){
10              dout.write(data);
11          }
12          dout.writeUTF("contacts");
13          dout.writeInt(contacts.size());
14          for(int i = 0; i < contacts.size(); i++){
15              byte[] contact = ((Persistent)contacts.elementAt(i)).persist();
16              dout.writeInt(contact.length);
17              if(contact.length > 0){
18                  dout.write(contact);
19              }
20          }
21
22          dout.flush();
23          return bout.toByteArray();
24      }
25
26      public void resurrect(byte[] persistent) throws IOException,
            PersistenceException{
27          ByteArrayInputStream bin = new ByteArrayInputStream(persistent);
28          DataInputStream din = new DataInputStream(bin);
29
30          String className = din.readUTF();
31          if(!className.equals(this.getClass().getName())){
32              throw new PersistenceException("Trying␣to␣ressurect␣wrong␣class!");
33          }
```

```
34
35          if(din.readUTF().equals("me")){
36              me = new BusinessCard();
37              int length = din.readInt();
38              byte[] data = new byte[length];
39              din.read(data);
40              me.resurrect(data);
41          }
42
43          if(din.readUTF().equals("contacts")){
44              int numberOfContacts = din.readInt();
45              for(int i = 0; i < numberOfContacts; i++){
46                  int contactSize = din.readInt();
47                  byte[] contact = new byte[contactSize];
48                  din.read(contact);
49                  BusinessCard contactCard = new BusinessCard();
50                  contactCard.resurrect(contact);
51                  contacts.addElement(contactCard);
52              }
53          }
54
55      }
```

15.2 The PersistenceManager Class

When you have created the persistent objects for your MIDlet, Peer2Me provides an interface for storing and retrieving them from the *RecordStore*. Through the static methods *updateOrStore* and *retrieve* in the *PersistenceManager* persistent objects can be stored and retrieved with an associated string key.

For further information on the PersistenceManager and the Persistent interface refer to the Javadoc on the Peer2Me website [1].

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

Part V

Testing

Chapter 16

Scenario Testing

In order to examine whether the chosen technologies and the Peer2Me framework are suitable for writing networked collaborative applications for mobile phones, we conducted several scenario tests. These tests tried to enact the scenarios on which the applications are based. Enacting the scenarios as accurately as possible gives a good measure of the suitability of the Peer2Me framework. The value of such collaborative applications for end users will then also be tested. An evaluation of the aspects regarding the Computer Supported Cooperative Work (CSCW) values for end users of these applications will be discussed in Chapter 19.3 in the discussion part of this report. To ensure objectivity, we got a few students at our department to perform the testing while we ourself functioned as observers to be able to document the results and receive comments from the testers. Each scenario test will now be described separately.

16.1 Business Card Exchange

This scenarios takes place at a large conference. Unfortunately we do not have any large conferences to visit where we can test the application. As a replacement we will use the university's cafeteria at lunch time. The cafeteria is the closest we get to a conference, since it shares a lot of properties with a conference:

- A large number of people are gathered in the same area.
- A large number of the people in the area have mobile phones with Bluetooth quite randomly switched on or off.

The test will be conducted by two people, Person 1 and Person 2, walking into the cafeteria at lunch time, meeting and using their phones to exchange business cards. After the transaction completes, Person 1 will leave and a third person will enter. Person 3 will approach Person 2 and will then exchange his business card with Person 2. Person 2 does not notice that this transaction takes place. When Person 2 finishes his lunch, he walks back to his office and sits down and browses his available business cards. He will then find both the business cards from Person 1 and Person 3. This test case can be formalized into three different steps:

Test Step:	Result:	Comment:
1	Success	Person 1 successfully downloaded Person 2's business card.
2	Success	Person 3 successfully downloaded Person 2's business card.
3	Success	Person 2 found both the business card from Person 1 and Person 3 on his mobile phone after the test had completed.

Table 16.1: Test result of the Business Card Exchange scenario.

Figure 16.1: Two people testing the business card scenario in the cafeteria.

Step 1: Person 1 exchanges business card with Person 2.

Step 2: Person 3 exchanges business card with Person 2.

Step 3: Person 2 looks through his available business cards.

16.1.1 Test Results

The test completed with no problems at all. Time is not an issue for an exchange like this since the participants of the scenario are collocated over time due to e.g. a common interest. Because of this, the communication between the devices had more than enough time to complete before they moved apart. The test results are summarized in Table 16.1. Figure 16.1 shows Person 1 and Person 2 exchanging cards.

16.2 PAN Instant Messaging

This scenario takes place during a lecture at a university. In order to test this scenario we will perform the tests in a lecture hall. The hall will be empty to make sure that no real lecture is

disturbed by our testing. This scenario test involves three different persons. The first person, Person 1, enters the lecture hall, sits down and turns on his PAN Instant Messaging applications making himself available for other users. After a while another person, Person 2, enters the lecture hall, sits down, starts the application and searches for other users. Person 1 is found and the they start a conversation. After a while another person sitting in the lecture hall, Person 3, also starts his Pan Instant Messaging application. Person 2 initiates a new search after devices, finds Person 3, and allows him to join the chat group. The three persons then exchange instant messages for a while. Suddenly, Person 1 disconnects and leaves the lecture hall. Person 2 and Person 3 continues the conversation for a while before disconnecting and leaving the lecture hall. This scenario test can be described as a list of test steps:

Step 1: Person 1 enters the lecture hall and turns on his PAN Instant Messaging application.

Step 2: Person 2 enters the hall, sits down and starts a search after other devices.

Step 3: Person 2 finds Person 1, connects to him and starts a conversation with him.

Step 4: Person 2 and Person 1 exchange messages for a while.

Step 5: Person 3 enters and starts his application.

Step 6: Person 2 searches for other devices, finds Person 3 and allows him to join the group.

Step 7: Person 1, Person 2 and Person 3 exchange messages for a while.

Step 8: Person 1 disconnects and leaves.

Step 9: Person 2 and Person 3 exchanges messages for a while.

Step 10: Person 2 and Person 3 disconnect and leave the lecture hall.

16.2.1 Test Results

As seen in Table 16.2 all the test steps completed successfully. Despite of that we still noticed a few things during the test and we also got a few comments from the scenario testers. A picture from the scenario testing of the PAN Instant Messaging application is shown in Figure 16.2.

It is important to note that the transfer speed of the Bluetooth technology seems more than good enough for such applications and scenarios as PAN Instant Messaging. The handling and parsing of messages inside the framework core is also a fast process which resulted in all messages being delivered and displayed in real time. Because of this we can truly conclude that the non-functional requirement about message transfer speed and real-time interaction, described in Chapter 7, has been fulfilled.

The routing protocol seems to scale pretty good for a small amount of devices. This protocol should be tested in a larger piconet (more than 3 devices) if more test phones are made available after this project has ended. In an application like PAN Instant Messaging there will be a lot of work conducted by the master node in the chat group. The master will have to route messages between every node in the group. This could make this phone loose battery power quite fast. A group management mechanism for changing master in the group during run-time could be applied

Test Step:	Result:	Comment:
1	Success	
2	Success	
3	Success	
4	Success	When the two devices had been connected and the conversation had started the exchange of messages happened in real-time. The transfer speed of the Bluetooth technology was more than good enough for this application
5	Success	Person 3 thought that it was a bit strange that he had to wait for Person 2 to find him in a new search before he could request a join to the group
6	Success	It seemed like there was no delay in the transfer of messages even if there were three nodes communicating. All messages appeared immediately on the receiving devices after they were sent from another device.
7	Success	Interaction happened in real-time.
8	Success	When Person 1 disconnected, he was automatically removed from the group. The two other nodes were then informed about this.
9	Success	
10	Success	

Table 16.2: Test result of the PAN Instant Messaging scenario.

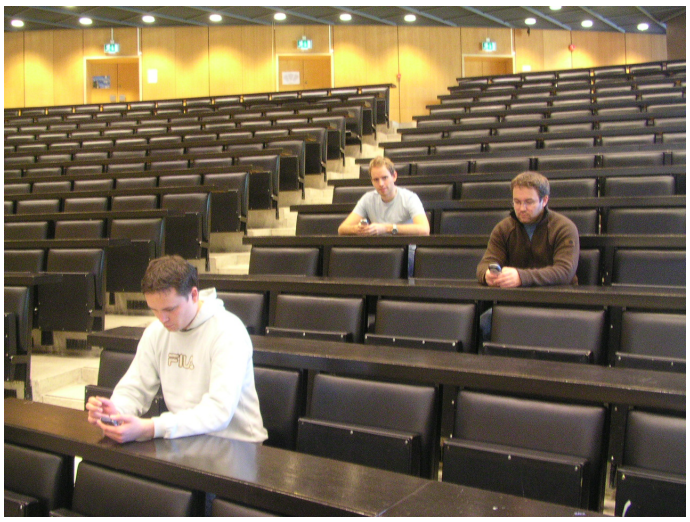


Figure 16.2: A picture from the scenario testing of the PAN Instant messaging application.

in order to not drain the power of a single master node. This is further described in Chapter 21 about further work.

During Test step 5, Person 3 thought that it was a bit strange that he had to wait for Person 2 to find him in a new group search before he could request a join to the group. This is a result of the nature of the Bluetooth piconet topology and specification. Slave nodes cannot request a connection to a master node. A slave node has to make itself available and then wait for a master to discover it and initiate a connection to it. This network paradigm influences in some degree how such applications as PAN Instant messaging have to be designed.

16.3 Converging Top Ten List

This scenario takes place at a random place where two people encounter each other by chance. In order to test this scenario we used the hallway outside our office at the university. The test cases involves two persons that are running the Converging Top Ten List application on their mobile phones. They both have registered a list of beer prices in their application. The test cases will cover different kind of encounters between these two. Each encounter will involve the exchange of the lists stored on each mobile phone.

We will divide the chance encounters into three different categories. These categories are:

1. Both devices moving in opposite directions. Encounters like these are very brief and give the shortest amount of time for the devices to connect and exchange their top 10 lists. This is equivalent to two people passing each other on the street or in the hallway.
2. One device standing still, the other passing by. This class of encounters last somewhat longer than the previous, but still yield a rather short amount of time to complete the communication between the devices. This is equivalent to one person standing still e.g. looking at a storefront display or a billboard while another person passes him.
3. Both devices halting at the same place. This class of encounters can last a long time giving the devices more than enough time to exchange their lists. This is equivalent to two persons standing still at the same place, conversing, waiting to cross the street or looking at the same product in a store window.

In order for the test to be complete and exhaustive we will test each of the three classes of encounters. The scenario test therefore will consist of three different test cases:

Case 1: Person 1 and Person 2 walking through the hallway in opposite directions.

Case 2: Person 1 standing still by the billboard in the hallway, Person 2 walking past him.

Case 3: Person 1 standing still by the billboard in the hallway, Person 2 walking up to the billboard and stop for 2.5 minutes.

Figure 16.3 shows a graphical representation of the three classes of encounters. P1 and P2 denotes Person 1 and Person 2 respectively. Because Bluetooth is used as network medium, each person

will have a communication range of about 10 meters. It is clear that the time at which the devices will be within communication reach of each other vary a lot from case 1 to case 3. The walking speed of the two persons will be crucial to whether or not the devices will have time to discover each other and exchange the necessary messages.

16.3.1 Test Results

When we started this scenario test, we soon realized there was no point in just conducting the three test cases and then just register if the test failed or not. The time interval when the devices actually can discovery each other and exchange information is very short, and the results of these test cases therefore highly depends on the time it takes to search for and discover other devices. We decided to test the discovery and connection speed of the framework and the Bluetooth module by writing and testing a simple test application. This application simply just searches for other devices, starts a handshake and connects to the appropriate available devices. The application includes time measurement functionality provided by J2ME by using the *Date* class. The testing was performed in an environment where no other Bluetooth devices could appear. This was done to avoid disturbance and interference with other devices and thereby reduce the variation of the test results. The test results are therefore the most optimal discovery times that can be achieved and in a crowded environment these processes might take longer time.

In Appendix A, we have included empirical data from the group search with each of our three test phones. The results varied from 18.3 seconds(s) to 25.4s. The results varies between the different phones because they differ in resources as CPU, memory, etc. and because the manufacturers has implemented the Bluetooth API differently. The best average was 20.2s (20239.3 milliseconds). The time used by the application to transfer the actual data after a connection has been established is so short that we will ignore it in the following calculations.

The formula for calculating distance traveled in a given time frame when traveling at a given average or constant velocity is:

$$d = v * t.$$

d: distance
v: velocity
t: time

This means that they have to walk slow enough for the handshake to complete before they have walked out of each others communication range, which gives us the unequality $d < v * t$ where $t = 20.2s$. The value d will vary depending on the walking pattern but relies on the communication range of Bluetooth which is 10 meters. This transforms into $v < \frac{d}{20.2s}$, which gives is a way to calculate the maximum walking velocity depending on the distance available. We will now use this formula to calculate the maximum walking speed Person 1 and Person 2 can have in case 1 and case 2. According to [58], the average walking speed of a normal pedestrian is somewhere between 1.2 and 1.4 meters pr. second. Comparing this to the calculated max walking speed for each of the cases gives us a reliable way of predicting whether or not the test will fail or succeed. We will now calculate the maximum walking velocity for each case and evaluate this.

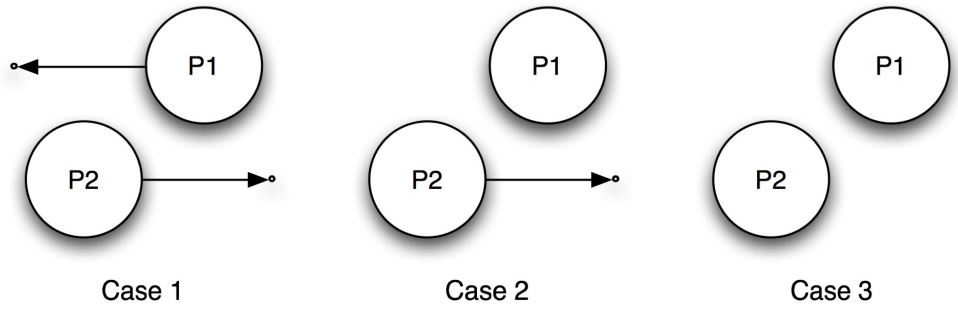


Figure 16.3: The three classes of chance encounters.

Test Case:	Result:	Comment:
1	Failure	According to the calculations done in this section (Section 16.3.1) this test case fails.
2	Failure	According to the calculations done in this section (Section 16.3.1) this test case fails.
3	Success	

Table 16.3: Test result of the Converging Top Ten List scenario.

Case 1: If we suppose that both Person 1 and Person 2 walk with the same speed in case 1, both will have to walk 10 meters before they get beyond each others communication range. This gives us a maximum speed of $v < \frac{10m}{20.2s} = 0.495m/s$ which is far below the normal walking speed. This means that unless we walk at less than half speed this test will probably fail.

Case 2: If we suppose that Person 1 is standing still and Person 2 is walking with a constant speed in case 2, Person 2 will have to walk 20 meters before he gets beyond communication range of Person 1. This gives us a maximum walking velocity of $v < \frac{20m}{20.2s} = 0.99m/s$ which is closer to the normal walking speed than in case 1, but still lower. This test will probably fail if we walk at normal speed but will have a chance to succeed if we drop our pace to about 25 - 30 % of the average.

Case 3: In this case the two persons stand still nearby each other in 2.5 minutes. This is more than enough time for the discovering and connection of the devices and the data exchange.

We verified that the application ran on the mobile phones and from that calculated whether the tests would succeed or not through the use of the above formulas. The test results are presented in Table 16.3.

Chapter 17

Developer Testing

As mentioned in Chapter 2.2.2, describing the empirical approach of this thesis, we wanted to conduct an empirical experiment to gather data related to Peer2Me and receive objective feedback from others. We decided to arrange a developer workshop to examine the usability and usefulness of the framework and to test how Peer2Me impacts on the development of mobile ad hoc applications. We managed to get 7 developers to participate. The workshop was divided into three parts: Education, development and evaluation. An overview of the workshop, the different parts and the results are described in this chapter. Measurement data and statistics gathered according to our measurement plan are also given.

17.1 The Education Session

In order for the participants to gain an understanding of how to develop applications using the Peer2Me framework, we started the workshop with an educational session. First, we gave a 20 minute overview lecture explaining the purpose of the workshop and introducing the framework to the participants, see Figure 17.1. After the lecture, the participants were given time to go through the Peer2Me development guide included in this report as Chapter 14. We wanted to test the usability and usefulness of the development guide so we were not available for guidance or questions during this part of the workshop. In order to measure the framework's complexity and degree of learnability, each participant could individually decide when they were ready to go on and start the next session, the development session. We noted the time each person spent on reading and learning about the framework.

17.2 The Development Session

In the development session, we gave each participant an application to finalize. We had prepared a simple chat application that was missing everything that had to do with network communication and P2P infrastructure. The task given to the participants was to have them finish the application by using functionality offered by Peer2Me. The task description is included in Appendix B.

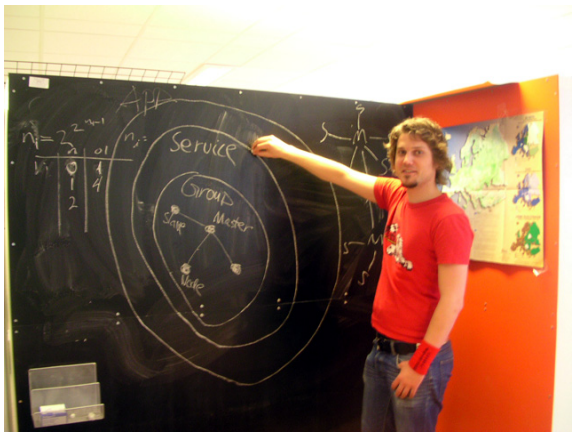


Figure 17.1: Michael is explaining the domain concepts of Peer2Me.

Figure 17.2: Two of the participants working with the programming exercise.

Our reason for handing out a half-way completed program was that we did not want the developers to spend time and effort on problems related to GUI and MIDlet programming. This would have blurred the real focus of the session, the Peer2Me framework. Two of the participants working with the programming exercise can be seen in Figure 17.2.

17.3 The Evaluation Session

We conducted an evaluation session after the participants had finished the programming exercise. This session started out with a questionnaire that all participants filled out. This questionnaire can be found in Appendix D. In this section we will list the main conclusions that we have drawn from the results of this inquiry.

17.3.1 Results From the Questionnaire

The participants were asked to define their background experience with respect to general Java development, years of experience writing applications and experience with Bluetooth and J2ME.

Statement:	Agreed (%):
The domain concepts of Peer2Me are easy to understand	100 %
The concept Framework is easy to understand.	100 %
The concept Message is easy to understand.	100 %
The concept Network is easy to understand.	100 %
The concept Node is easy to understand.	85 %
The concept Group is easy to understand.	71 %
The concept Service is easy to understand.	51 %
The domain concepts simplifies the problem domain.	71 %
The development guide helped me through the exercise.	100 %
The development guide is easy to read.	57 %

Table 17.1: Statements evaluated by the workshop participants.

The results were as follows:

- All the participants described their java and programming skills as good or professional.
- The participants had 5-10 years general programming experience and 3-5 years of java experience.
- None of the participants had developed a mobile or a Bluetooth application before.
- None of the participants had ever used J2ME before.

The participants were asked to evaluate and agree/disagree to certain statements about the Peer2Me concepts and the developer's guide. The results of this is shown in Table 17.1. As we can see the developers thought the Peer2Me concepts were simple to understand and that the developer guide helped them in writing the exercise.

We also received some suggestions on how we can improve our development guide. Many of the participants suggested that we should explain more about the differences between a master and a slave node. There were also suggestions regarding the readability of the guide, like removing unnecessary javadoc and showing longer snippets of code. Some of the participants also wanted more code examples in the guide.

The developers were also asked to rate the parts of the exercise in to different dimensions. The first was, "Which part of the exercise was the most difficult to understand" and the second was "Which part of the exercise was the most time consuming". The results are shown in Figure 17.3 and 17.4 respectively.

Almost all participants thought it was difficult to understand the difference between a master and a slave node. They also thought it was difficult to understand that they were going to write code for both the master and the slave, and not just one of them. Some of the developers thought that it was very difficult to debug the code they where writing. All the participants had problems with one or more NullpointerException that occurred while testing and they thought it was very difficult to find the cause of the exception. Some of the participants did not understand the relation between a group and a service. A few of the developers also complained about that information regarding the RunningService method was not included in the development guide.

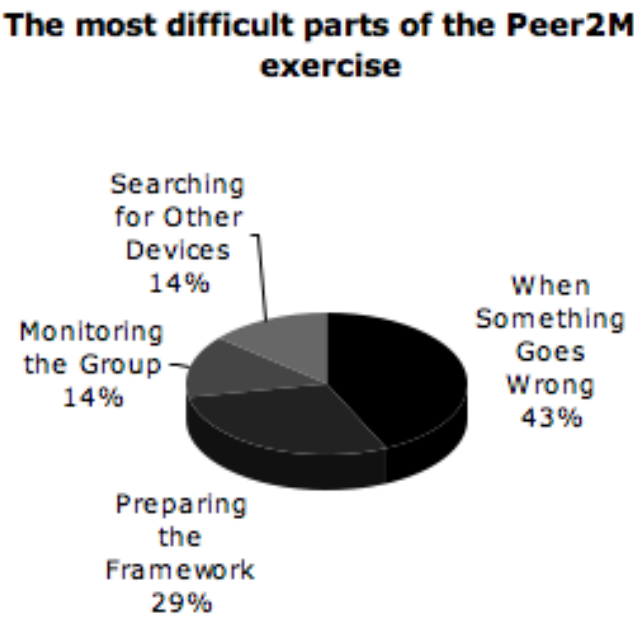


Figure 17.3: The most difficult parts of the exercise as rated by the participants of the developer workshop.

We also asked the developers how they thought Peer2Me could improve the developing process when writing mobile collaborative applications. Most of the participants thought it was difficult to answer these two questions because they had no experiences with developing similar kinds of applications to refer to. The answers we got were:

- 71% of the participants thought that Peer2Me speeds up the development of collaborative mobile applications.
- 71% thought that Peer2Me clarifies the domain of mobile collaborative applications

17.3.2 Post Morten Analysis

We ended the evaluation session with a lightweight post mortem review of the session itself and the framework in general, see Figure 17.5. This was conducted as a brainstorming session, inspired by a post mortem review method described in [18]. We asked a few questions and started a discussion for each of them. We included this session in addition to the questionnaire to try to catch ideas from the developers that they did not get a chance to express in the questionnaire. We wanted to use their knowledge and experiences to evaluate the framework and maybe come up with some new ideas. The questions we asked were:

- Do you think Peer2Me will be useful for developers?

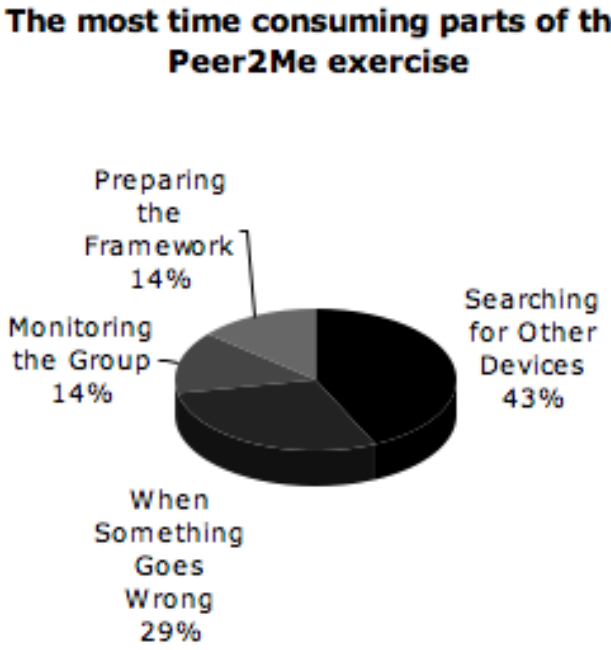


Figure 17.4: The most time consuming parts of the exercise as rated by the participants of the developer workshop.

  - The participants thought that it was difficult to answer this question because none of them had ever developed a mobile collaborative application before.
  - One of the developers meant that developing with Peer2Me should be as easy as developing ordinary stationary network applications.

- In what way can Peer2Me be improved?
  - One developer suggested that we should try to gather similar functionality into methods. For instance, method calls that often occur in sequence could be generalized into one method.
  - Almost everyone thought that the usage and the descriptions of the master and the slave node concepts should be made clearer.

- Other comments:
  - Everyone agreed that the way J2ME and Wireless Toolkit handles exceptions complicates the development of applications.
  - Some of the participants thought that the programming exercise would have been easier if they had previous experience with the Wireless Toolkit.
  - Several participants remarked that they thought it was a lot of fun developing applications with Peer2Me.
  - One of the developers thought that some methods in the framework should have been named more intuitively.

Figure 17.5: A picture from the brainstorming session.

17.4 Measurement Data and Statistics

In this section we will list and discuss the measurement data and the statistics we gathered during the workshop. This data was gathered according to our measurement plan defined in Chapter 2.2.2.

From Table 17.2, we can see that almost all the developers managed to do all the sections of the programming exercise. These results are far above what we had predicted. We also found that the programming session was very helpful to increase the understanding of the concepts behind the framework and when the development session was finished, the participants remarked that their knowledge level had increased. Because of this it seams like Peer2Me has a steep learning curve[1], which means that the competence increases quickly over time while learning and using the framework. This again increases the usability and usefulness of the framework. Table 17.2 also shows how much time each developer spent on each part of the exercise, we will get back to these results in Chapter 19.

The developers also gained a lot of knowledge by asking a lot of questions during the programming exercise. We answered these questions and straightened out the arising misunderstandings. We also kept statistics over which questions that were asked at what time. This statistic will indicate what parts of the exercise that was the most difficult for the developers and could also say something about the learning curve of the framework. The number of questions asked per developer during the different sections of the exercise are shown in Table 17.3. This table shows that the majority of the questions that were asked were asked in the first half of the exercise. This means that the developers learned as time went by and in the end they could work more independently than in the beginning. This could also mean that the most central concepts of the framework, that

[1]The phrase "a steep learning curve" is today become a cliche and is often misused by referring to something that is difficult to learn. This is wrong, because if a learning curve is steep, this means that for smaller increments of time, larger gains in learning are accomplished, which is the opposite of what the cliche says. By a steep learning curve we mean something positive and that the competence is increasing quickly over time.

Developer Number:	Time Spent On Training:	Time spent on exercise:	Sections Finished:
1	35 min	100 min	All, except nr. 5
2	45 min	90 min	All, except nr. 5,6
3	43 min	88 min	All
4	42 min	83 min	All
5	42 min	73 min	All
6	45 min	90 min	All
7	37 min	83 min	All
Average	41.3 min	86.7 min	-

Table 17.2: Time measurement from the developer workshop.

Developer Number:	Section 1	Section 2	Section 3	Section 4	Section 5	Section 6	Section 7
1	1	1					1
2							
3							
4	1		1	1			
5	2	1	1	1			
6	1						
7	1		1	1			

Table 17.3: Table showing how many times the developers asked for help during the different sections on the programming exercise.

where used in the early parts of the exercise, are the most complex concepts to understand. More precisely it seems like the master and slave node concepts are the most difficult to understand.

17.5 Summary

We will here list what we think was the most important suggested improvements to the framework or its documentation from the developer workshop:

- The main idea and the way of using the master and the slave node concepts should be explained more clearly and in more detail in the development guide.
- Collect and gather similar functionality into more high-level methods. Reduce number of methods in the interfaces that can be used by the application developer.
- The development guide could be made more readable.

Part VI

Discussion

Chapter 18

Encountered Problems

This section will describe the problems that we have encountered during the project. A more in depth evaluation will be given in Chapter 19.

18.1 Mobile Phones and J2ME

A problem regarding the mobile phones is that the same application looks radically different on different mobile phones. The mobile manufactures implement J2ME functionality quite different. Designing a graphical user interface that has good usability on one phone does not ensure that the same interface is usable on another phone.

Another problem is multitasking and the lack of mulittasking. The Nokia 6600 is apparently able to multitask applications and has a task manager for raising applications that were previously placed in the background. The Sony Ericcson p900 seems to be able to run applications in the background, but we have been unable to find a suitable way of getting applications, that have been placed in the background, back to focus. Siemens S65 does not seem to have any form of multitasking at all.

18.2 Bluetooth

Bluetooth devices running as masters can not accept incoming connections from other masters. In addition, slaves can not function as slaves in more than one piconet at a time. This has severe impacts on creating scatternets with bluetooth enabled devices. It also has an impact on our scenarios in the Hybrid and Auto categories. When devices are doing continuous search for other devices, they might not find everyone that is within reach since they are already connected in another piconet.

There are variations on how many Bluetooth connections the various phones support. The Sony Ericsson p900 only supports one connection at the time. Siemens S65 seems to support 3, we know for sure that it supports 2. The only phone that seems to fully implement the Bluetooth is the Nokia 6600. According to the implementation on the phone it supports 7 connection, e.g.

a complete piconet. Since we only have three phones for testing we only know for sure that it supports 2 connections. This variety in the Java APIs for Bluetooth Wireless Technology (JABWT) implementations put restrictions on which phones can be used for what purposes and how. These different properties of the different JABWT implementations are not public documented which makes the whole development process quite explorative.

The Sony Ericsson p900 has a faulty JABWT implementation. When returning from a device discovery process the p900 always return with the value DEVICE_DISCOVERY_ERROR, even if devices have been found properly. This is apparently due to a mixup of two return values in Sony Ericsson JABWT implementation. Sony Ericsson have been notified of the problem, but have been unable to come up with a fix for it. Today the problem is present in all the high-end Sony Ericsson mobile phones with JABWT support.

The different JABWT implementations are slow and implements highly varying disconnection detection times. How long it takes for a phone to detect that it has been disconnected from another phone varies a lot between the different phone models. In general it takes a lot of time, from 5 to 15 seconds.

Chapter 19

Evaluation

This chapter gives an overall evaluation and analysis of the work that have been conducted and the results of this project. We will first evaluate the most technical issues related to the framework and the different technologies used. We will then do an evaluation of the empirical work by performing the interpretation phase of the Goal Question Metric (GQM) method and by evaluating the Peer2Me example applications.

19.1 Technical evaluation

This section looks at the technologies used and the technical results produced. Through this evaluation we will answer the following research questions:

- Is it technical possible to develop and implement a framework like Peer2Me on mobile phones?
- Are mobile phones together with J2ME a suitable technical platform for mobile collaborative applications?
- Is Bluetooth a suitable technology for mobile collaborative applications?

19.1.1 Framework

The requirements we found in [37] and updated in this report is completely covered in our design. Our implementation of the Peer2Me covers all but two requirements completely, these two are partially covered.

The Peer2Me implementation consists of more than 1100 lines of code, but is still kept within the size limitations of mobile applications. The resulting package is well within the limits of the storage capacity of the mobile phones of today. Our choice of an explorative development method has allowed us to continuously improve Peer2Me and quickly discover and fix encountered problems.

When it comes to the functionality embedded in the Peer2Me framework, we have utilized the technologies to the full, trying to work around their limitations and shortcomings. For instance, the master-slave network topology of a Personal Area Network (PAN), where a master connects to a number of slaves, instead of a slave connecting to the master, was at first a bit hard to grasp. This topology also influenced how we designed the group management mechanism in Peer2Me. The developer workshop also confirmed that the concept of master and slave is one of the most difficult concepts to grasp.

19.1.2 Mobile Phones and J2ME

Mobile phones work well with the problem domain. They are lightweight, a large number of people own a mobile phone and the phones are almost always on and connected. The Java 2 Micro Edition platform (J2ME) also enables us to deploy the Peer2Me framework on a number of phones due to the portable nature of J2ME.

Despite of that, mobile phones are not without shortcomings. The J2ME implementations on phones produced by different manufacturers vary a lot, especially when it comes to optional functionality such as Bluetooth, etc. The Mobile Information Device Profile (MIDP) 2.0 also limits the applications access to certain features of the phones operating system and resources. With MIDP 3, a lot of such technical issues will be improved, allowing applications to start at boot time and run in the background while the phone is on.

19.1.3 Bluetooth

The most serious problem with the Bluetooth technology for use in mobile collaborative applications is the slow processes of discovery and searching for other devices. The currently available Bluetooth enabled devices use, at best, 10.24 seconds to discover other devices in their proximity, [24, 64]. This is a long time if devices are moving or communication time is limited for other reasons. The next generation of Bluetooth devices, that will conform to the version 2.0 of the Bluetooth specification, will probably offer radically shortened discovery times.

In spite of the discovery time, Bluetooth has been proved to work for moving devices as described in [59]. Welsh, et.al. describe an experiment where they actually perform their tests with Class 1 Bluetooth devices with a range up to 100 meters and speeds above 20 km/h (5.56 m/s). This should scale down and prove that it is possible to use Bluetooth for exchange between Class 2 devices (mobile phones) with a range of 10 meters at walking speed (1.4 m/s). During our scenario testing of the Converging Top Ten List application, described in Chapter 16, we found the best test mobile phone to perform discovery processes in an average of 20.2 seconds.

It seems that a lot of the problems connected to Bluetooth's long discovery times on mobile phones are related to the J2ME implementation of the Bluetooth APIs (JABWT). Bluetooth tools and software not implemented in J2ME, produce a list of discovered devices which is updated as new devices are found. These devices can be connected to instantly if desired. With JABWT, connections can not be made until the discovery process has completed, making the response time much higher. We learned this when trying to parallelize the discovery process in our Bluetooth module to reduce the response time of a search after devices. As stated in Chapter 8.5.2, the

implementation of the JABWT in the mobile phones do not allow this discovery to be parallel. Hopefully, this will be fixed in future releases of JABWT.

In Chapter 21, we discuss some improvements that can be made to the Bluetooth specific parts of Peer2Me based on experiences from other research projects.

In spite of its shortcomings, Bluetooth remains the most promising Personal Area Network (PAN) technology. Its transfer rate is more than adequate for the scenarios we have discussed in this report, enabling real time communication between nodes. The number of Bluetooth enabled phones is high and rising. The APIs are complete although the implementation has shortcomings.

19.1.4 Development Platform and Environments

From the developer perspective, J2ME is a bigger challenge to develop applications on than the Java 2 Standard Edition (J2SE). The reason for this is that the developer tools for J2ME is not yet as mature as the Integrated Development Environments (IDEs) of J2SE. As an example, tracing exceptions is difficult with the tools we have used since the stacktraces produced lack a lot of information compared to the stacktraces produced in J2SE.

In addition to this, the emulators currently available do not have the same behaviour as the mobile phones. The phones run the applications with limited resources such as CPU capacity and memory, and the emulators run the applications with close to unlimited CPU capacity and memory. This causes a mismatch between how applications behave on the emulators and mobile phones. This complicates the development process when it comes to testing and debugging. Debugging and the search after errors is also very time consuming on the mobile phones.

19.2 The Interpretation Phase of the Goal Question Metric method

This section will evaluate the following research question:

- Will developers benefit from using Peer2Me when developing mobile collaborative applications on mobile phones?

To be able to answer this question we described it as two different goals, see Section 19.2.1 and Section 19.2.2 below. In Chapter 2.2.2 we used the Goal Question Metric (GQM) method to break down these goals into questions and metrics and we will now perform the interpretation phase of the GQM method to answer these questions. Some of the metrics describes quantitative data that was gathered and measured during the developer testing described in Chapter 17. The other metrics describes data that could not be measured or gathered through experiments. These data will now be estimated qualitative according to our own and others experiences and used to answer the defined questions together with the quantitative data. When the questions are answered, we will use these answers to evaluate if we have reached the two main goals. This analysis constitutes the interpretation phase of the GQM method.

19.2.1 Evaluation of Goal 1

In Chapter 2.2.2 we defined the first goal of our empirical work by using the GQM template as follows:

> *Analyze* Peer2Me
> *for the purpose of* Deciding if it will be easier to develop applications using the framework
> *with respect to its* Usability, usefulness and effectiveness
> *from the point of view of the* Developers
> *in the context of* Mobile collaborative application development.

We will now evaluate and answer each question from the GQM three that relates to this goal. In the end we will conclude upon if we have reached the goal or not.

QUESTION 1: Is it easy to understand the concepts of Peer2Me?

Metric: Time spent on Peer2Me training.

Data measured: 40 minutes.

Peer2Me is constructed to simplify the domain and should therefore be more easier to learn than a specific technology. In the developer workshop the participants spent in average 41.3 minutes on training. This training was very effective and covered so much that 71% of the participants finished all sections of the programming exercise before the time exceeded. 71% of the developers also thought that the domain concepts simplified the problem domain and the same number of developers also thought that Peer2Me clarifies the domain of mobile collaborative applications as a whole. The developers though thought that the concepts of a master and a slave node were a bit difficult to understand. We believe the main reason for this is that developers nowadays are so used to the client-server paradigm that it takes some time to understand an alternative to this paradigm.

Answer: Yes, the concepts of Peer2Me is more or less easy to understand, but the learnability of the framework depends on the level of motivation and the ability for adopting a new way of thinking about networked applications.

QUESTION 2: Is it easy to understand the concepts of a wireless Personal Area Network technology as Bluetooth?

Metric: Time spent on learning the concepts of Bluetooth.

Data estimated: 8 hours.

From our own experiences with the Bluetooth technology, we have learned that it takes a while to grasp the concepts of it. In the prestudy phase of our in depth study project in 2004 we developed a prototype application based upon Bluetooth technology before we started developing the framework itself. We spent three days making this applications. The first day we used to understand and learn about the concepts behind the technology. Then we spent two days on the actual development of the example application. During these two days we spent more or less one whole day working on the Bluetooth specific parts and another day working on other components related to J2ME.

To verify our estimates we have contacted one of the major contributors to the Java Specification Request (JSR) 82, the Java APIs for Bluetooth Wireless Technology (JABWT), a company called Rococo Software. Rococo software is an active member of the JSR 82 Expert group and has developed and worked with Bluetooth for a long time. Rococo Software also offers a comprehensive collection of Bluetooth and wireless technology training courses. One of the training courses provides an introduction to the Bluetooth technology. This course aims to introduce the main concepts of the Bluetooth technology to the participants. According to [50], the duration of this course is one day and it requires no prerequisites. This verifies our estimate that it takes more or less one whole working day, about 8 hours, to learn the main concepts of the Bluetooth technology from scratch.

Answer: No, to understand the main components and concepts related to a wireless PAN technology, like Bluetooth, is a complex and time consuming task.

QUESTION 3: Is it easy to learn developing mobile collaborative applications with Peer2Me?

Metric: Time spent before mastering Peer2Me development.

Data measured: 130 minutes, about 2 hours.

Metric: Number of people that succeeds doing a programming exercise.

Data measured: 71% (5 out of 7 developers).

According to the results of the developer workshop the learnability of the Peer2Me framework is very good. It seems like the framework has a steep learning curve which means that the competence increases quickly over time while learning and using the framework. Much of this learnability depends on a well written and informative development guide. 100% of the participants of the developer workshop thought that the development guide, found in Chapter 14, helped them through the programming exercise.

Answer: Yes, it is easy to learn developing with Peer2Me, but the learnability depends upon a well designed development guide.

QUESTION 4: Is it easy to learn developing mobile collaborative applications with Bluetooth?

Metric: Time spent before mastering Bluetooth development.

Data estimated: Three working days, about 24 working hours.

As stated above we spent three days, starting from scratch without any significant knowledge, making a prototype Bluetooth application. According to [51], Rococo software also estimates that it takes about three days to learn mastering Bluetooth development. They offers a three days course that gives the participants a from scratch understanding of the concepts behind the technology, an introduction to the JABWT and introduction to a step-by-step approach to Bluetooth development resulting in that each participant develops a working application. This verifies our estimate on how much time that must be spent to master Bluetooth development.

Answer: No, it takes a lot of time and effort to learn developing with Bluetooth.

Conclusion

We have now answered and evaluated upon all the questions related to Goal 1. The answers and the evaluations have made us reach Goal 1 because we can now conclude that Peer2Me makes it easier to develop mobile collaborative applications. A summary of the answers collected from this section supports this conclusion:

- The concepts of Peer2Me are more or less easy to understand.
- To understand the main components and concepts related to a wireless PAN technology, like Bluetooth, is a complex and time consuming task.
- It is easy to learn developing with Peer2Me.
- It takes a lot of time and effort to learn developing with Bluetooth.

19.2.2 Evaluation of Goal 2

In Chapter 2.2.2, we defined the second goal of our empirical work by using the GQM template as follows:

> *Analyze* Peer2Me
> *for the purpose of* Deciding if it requires less effort to develop applications using the framework
> *with respect to its* Effectiveness and usefulness
> *from the point of view of the* Developers
> *in the context of* Mobile collaborative application development.

We will now evaluate and answer each question from the GQM three that relates to this goal. In the end we will conclude upon if we have reached the goal or not.

QUESTION 1: How much time does it take to develop an application with using Peer2Me?

Metric: Time spent developing

Data measured: 90 minutes

This time metric says something about how much time it will take to develop an application after spending time on training. The participants in our developer workshop spent an average of 86.7 minutes to complete a simple chat application after attending the education session. It is important to notice that the time spent on developing probably would have been longer if the participants did not receive guidance throughout the development. It is also import to state that the chat application made during the workshop was very simple providing only basic functionality.

Answer: Developing an application using Peer2Me requires a small amount of time.

QUESTION 2: How much time does it take to develop an application without using Peer2Me?

Metric: Time spent developing.

Data estimated: 8 hours.

We will again compare Peer2Me development to Bluetooth development. We spent two days only on development when making our example application in our last project in 2004 mentioned above. One of these days was spent on making the Bluetooth specific code using the JABWT. According to [52], Rococo Software is offering a one-day workshop to learn how to make Bluetooth applications using the JABWT. Another company called Teleca also offers Bluetooth specific training. According to [55], one of their courses lasts for 8 hours and provides an introduction on how to write applications based upon Bluetooth. The participants are guided through some fundamental steps and ends up with writing a simple chat application. These references together with our own experiences make 8 hours as an appropriate data estimate for this metric.

Answer: Developing an application without using Peer2Me requires significant more time than the opposite.

QUESTION 3: How much source code is produced when developing an application using Peer2Me?

Metric: Lines of code (LOC).

Data measured: 108 LOC (Simple chat application)

Data measured: 242 LOC (PAN Instant Messaging)

QUESTION 4: How much source code is produced when developing an application without using Peer2Me?

Metric: Lines of code (LOC).

Data measured: 586 LOC (BlueChat by Ben Hui)

To answer Question 3 and Question 4 we have to refer to some example applications. We have chosen to use a standard chat application as reference to compare how much source code that is produced with and without using Peer2Me. During this project we have developed two different chat applications. The most simplest application, called a Simple Chat Application, was developed as a solution to the programming exercise in the developer workshop. This was a very simple application and was made up out of 108 lines of code. A more advanced chat application, the PAN Instant Messaging application, described in Chapter 12, was developed as a full-scale Peer2Me example application, and ended up consisting of only 242 lines of code.

To compare the number of lines of code held by these applications to applications developed without using Peer2Me we had to find a similar chat application developed from scratch using Bluetooth. One of the most important and popular Bluetooth programming resources on the web is the site maintained by Ben Hui[1]. Ben Hui is one of the most experienced Bluetooth developers out there and he released an article in February 2004 [29], that explained the main concepts of J2ME Bluetooth programming. This article explained the different parts in a chat application, called BlueChat, developed by Ben Hui. This application has later become one of the most referred Bluetooth applications available. The BlueChat application provides a bit more functionality than the Simple Chat Application, but less functionality than the PAN Instant Messaging application. The BlueChat application consists of 586 lines of code, which shows that much more source code is produced when not using Peer2Me when developing applications.

Answer to question 3 and 4: Less source code is produced using Peer2Me when developing applications.

Conclusion

We have now answered and evaluated all the questions related to Goal 2. The answers and the evaluations have made us reach Goal 2 because we can now conclude that it requires less effort to develop mobile collaborative applications using the Peer2Me framework. A summary of the answers collected from this section supports this conclusion:

- Developing an application using Peer2Me requires a small amount of time.
- Developing an application without using Peer2Me requires significant more time than the opposite.
- Less source code is produced using Peer2Me than without using it.

[1] http://www.benhui.net/

19.3 The Peer2Me applications

This section will evaluate the following research questions:

- Is Peer2Me suitable for developing all categories of mobile collaborative applications described in the classification matrix?
- Are these kinds of collaborative applications useful for end users?

Peer2Me is a framework for making collaborative applications on mobile phones. These are collaborative applications that operates at the same place and in real time. The categorization matrix, described in Chapter 4, are defining three main categorizes of such applications: User, Auto and Hybrid. Will these kinds of applications be adopted by end users? Are these applications promoting face-to-face interaction or utilizing the advantages of persons being collocated? Since we in this project have developed one example application from each of these categorizes, we now want to do an evaluation of how suitable Peer2Me is for making these kinds of applications. By analyzing the scenario testing, described in Chapter 16, we will also include a discussion of how the different categorizes of applications and scenarios seem to affect human collaboration and how well they seem to function in the domain of CSCW.

The BEDD software suite, described in Chapter 5, offers similar kinds of applications as Peer2Me to end users in Singapore. Services like proximity dating, buying and selling and chatting were quickly adopted by over 1500 users during the first two months in the capital of Singapore in the spring of 2004. Although these services were not for free, people seemed interested in such kinds of services on mobile phones. This shows the potential for these kinds of applications. The BEDD Corporation does not have any information about how many users they have today. The BEDD software might had been adopted more widely if it was available not only on Symbian phones and if it had used another business model.

19.3.1 User - Explicit User Interaction

These kinds of applications require user interaction. The user has to explicitly trigger the collaboration activities, start the information exchange or request a service. The PAN Instant Messaging application, described in Chapter 12, is an example of such an application. As a tool for spontaneous ad hoc group interaction there is no doubt that this kind of application is valuable. During the scenario test the collocated users communicated in a simple and effective way. This was done without using a cellular network, like GSM, and thereby also without being charged from a network vendor. Being able to form and communicate within an ad hoc chat group of physically collocated people can be suitable and effective in many different situations and environments. These are for instance communication during a conference or lecture, chatting with strangers on a bus, situations requiring exchange of secret information without using voice and so on.

These kinds of applications are not that depending on fast device and connection times and the infrastructure that is handled by Peer2Me covers a lot of the functionality required by these types of applications. This makes Peer2Me very suitable for making such applications.

19.3.2 Auto - Automatic Collaboration

These applications involve automatic collaboration between devices. The application is responsible for initiating communication between devices on behalf of the user. The user stores a profile that defines how the application should act with respect to other devices and available services. These applications will often be used to exchanging and spreading information in a population. The society today is overfilled with more or less useless information and it is a great challenge to find relevant and good quality information. It should be possible to exploit the fact that we every day move around and meet other people, familiar or unknown. Many of these people may have the same interests, share a common goal or need for information as you, but information of interest may only be in possession of a few of them. Applications that utilize automatic collaboration without requiring user interaction may be used to exchange such information.

We have made one example application from this category of applications, the Converging Top Ten List application. This is an application that can maintain and store top ten lists of product or service prices. These lists can then be spread around by automatic exchange to other people and then converge to the true top ten list. All the people in the population that use this application share a common goal. They want to get objective information about what products or services that offer the best prices. When these people move around and encounter each other, their physically collocation is exploited by the application to connect and exchange information. The top ten lists are compared, exchanged and thereby populated. We have not had enough resources to test this application in a large population to test its practical usefulness for a big crowd of end users.

The scenario testing of the Converging Top Ten List application involved testing if the technology that Peer2Me is based upon is possible to use when making such applications. The Peer2Me framework seems very suitable for making such applications. The big problem is the immature and poor implemented network technologies. In these kinds of applications the processes of discovering and connecting to other devices have to be performed fast because the people that carry the devices move around. The scenario test showed as mentioned earlier in this chapter, in Section 19.1.3, how the Bluetooth technologies in the different mobile phones limits the usefulness of the Peer2Me framework to make applications facilitating automatic collaboration. If the mobile phone vendors in the, hopefully near, future releases phones with faster and better implemented wireless technology, Peer2Me will be a valuable technology to use to develop such applications.

19.3.3 Hybrid - A Combination of User and Auto

These applications provide a combination of forced user interaction and automatic collaboration. There are mainly two different types of such hybrid applications:

Type 1: These are applications that automatically trigger collaboration that requires further user interaction. For instance: Automatic exchange and matching of music recommendations. When people with same taste in music meet, the application will alert the users and open for further user interaction and exchange of music.

Type 2: These are applications that are constantly running and that other users can connect to and download data from. An example of such an application is the Business Card Exchange application that we have developed in this project.

These kinds of applications can combine the best properties of the two first categories of applications, but they will also suffer from the technology limits of the automatic collaborative applications. The “Type 1”-applications are highly depending on the network technology in the same way as the “Auto” applications, but “Type 2” applications such as Business Card Exchange suit the framework very well.

Chapter 20

Conclusion

In Chapter 2 we posed several research questions concerning Peer2Me and its problem domain. Through our work described in this master thesis we have worked according to our method description with the aim of answering the research questions. The questions were answered through our project evaluation in Chapter 19 but we will now repeat the questions and summarize our results.

1. Is it technical possible to develop and implement a framework like Peer2Me on mobile phones?

   - Yes, we have developed the Peer2Me framework with applications and tested it on mobile phones.

   (a) Are mobile phones together with J2ME a suitable technical platform for mobile collaborative applications?

   - Yes, but there are currently a few shortcomings which hopefully will be resolved in the future.

   (b) Is Bluetooth a suitable technology for mobile collaborative applications?

   - Yes, but there are currently a few shortcomings related to the J2ME Bluetooth APIs.

2. Will developers benefit from using Peer2Me when developing mobile collaborative applications on mobile phones?

   (a) Will it be easier to develop applications using Peer2Me?

   (b) Will the development time be reduced when using Peer2Me?

   - Yes, tests have shown that training and development time is drastically reduced by using the Peer2Me framework when writing applications.

3. Is Peer2Me suitable for developing all categories of mobile collaborative applications described in the classification matrix?

- Yes, but the shortcomings in the J2ME creates a limit on how fast devices can move when running applications in the Auto category. This will probably be resolved in the future.

4. Are these kinds of collaborative applications useful for end users?

   - We did not have enough time to test the applications extensively on a user population and therefore lack the empirical data to conclude anything on this point, but our belief is that user can and will benefit from services offered through Peer2Me based applications.

We have improved and implemented a new version of the Peer2Me framework prototype and developed several applications that use the framework. The most noteworthy applications are the three example applications Business Card Exchange, PAN Instant Messaging and Converging Top Ten List, described in Part III (The Peer2Me Applications) of this report. Through empirical studies of developers using Peer2Me, we have found that there is a substancial benefit to be gained when developing applications both when it comes to understanding the domain concepts and learning to use the API and implementing the actual applications.

The Peer2Me framework is designed with a semi-layered structure in order to ensure modifiability and independence of network technology. The layers are the network module, the network interface and the framework. The bottom layer, the network module, contains all network technology specific parts of the code. The network interface provides an abstraction above this, making the top layer, the framework layer, completely independent of underlying network technology. In addition to this, there is a vertical package used by all layers, the domain package, that deals with all concepts central to Peer2Me such as node, group, service, etc. An application using the Peer2Me framework should only use objects from the framework layer and the domain package to ensure complete network independence. Well known design patterns are used in the design to ensure quality. Different kinds of communication protocols have been designed and implemented.

The Peer2Me framework is unique in the sense that we have been unable to find any other generic frameworks that cover the same problem domain as Peer2Me. We have not discovered any similar products implemented in neither Java nor any other native language. The closest thing we come to competitors are the BEDD software and the JSR 279: Ad Hoc Networking API. BEDD is a commercial software suite that can not be used by independent developers. The JSR 279 is still far from completion and it will probably be another two years before we see it supported on mobile phones. Peer2Me's uniqueness is further shown by the attention we have received from other universities worldwide. Through the project website[1] and conferences, we have been contacted by several researchers that are interested in our work, some of them wanting to use Peer2Me as a building block for their own research projects.

We have used an iterative approach for developing both the Peer2Me framework and the example applications. Through this iterative method we have been able to continuously monitor and evaluate the status of the implementation and discovering problems before or as soon as they appeared. The separate, but synchronized iterations used for each application gave us a way to test the framework step by step and improve the parts that had flaws. The total process has rendered

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

us a prototype which can be used for testing new scenarios and verifying the suitability of the technologies in the applicable scenarios.

Another aspect of using an iterative method is the learning process. Learning as you need the knowledge gives you hands-on experience and ensures that the learning curve does not grow to steep. Throughout this project we have learned a lot about formal research methodologies, structured analysis of problems and general object oriented design and development techniques. Designing and building a generic framework introduces new aspects and challenges when it comes to polymorphy and structure in the software. We have also gained more experience when it comes to debugging and fault analysis using monitoring tools. All in all we have enjoyed exploring the very interesting and growing domain of mobile collaboration and we hope that our work has been a contribution to the field.

Chapter 21

Further Work

Although Peer2Me in its present version works for testing scenarios and verifying the correctness of requirements, we have uncovered that the Peer2Me framework is not yet ready to be deployed with the end users. This chapter will discuss what needs to be done with Peer2Me concerning both short term and long term goals.

21.1 Short Term Goals

In this section we will describe the short term goals of the Peer2Me framework.

21.1.1 The Framework in General

The remaining requirements should be implemented in order to make sure that the Peer2Me framework is fully functional and works with all specified scenarios. The suggested improvements, regarding both the framework and its documentation, from the developer workshop, described in Chapter 17.5, should be considered designed and implemented.

The overall stability and reliability of Peer2Me should be a quite large focus to make sure that applications built on Peer2Me can be trusted by both developers and end users to actually work as expected and have a high degree of reliability. Today there are some problems connected to the disconnection of nodes.

Example applications that is made by the framework could be made more useful for a end user if he or she could access data stored in the native operating system of the mobile phones. This is information and data such as telephone numbers, notes, multimedia files, games and programmes. This kind of information should be accessable and integrated into Peer2Me applications and it may be suitable to integrate mechanisms and functionality for doing this into the framework core. Such integration may harm the security and privacy of the users, and solutions to manage these threats should be implemented. Security threats issues of the framework as a whole should also be evaluated.

21.1.2 Messages

Today, Peer2Me only supports messages with ASCII characters. In order for some of the scenarios to work properly, messages containing binary data, for instance images, should be supported by the framework. If this is done with the text message format, binary data will have to be Base64 encoded and decoded. Base64 encoding introduces a lot of overhead which is undesirable on mobile battery powered devices where low transfer times are crucial. When introducing a new binary message format, profits may also be gained by looking at the way messages are sent in the Bluetooth network module.

Extensible Markup Language (XML) may be used for describing messages. This ensures a more loose coupled communication between devices and much more readable messages. Using XML for describing messages enables the use of an XML parser to parse messages. This makes the parsing process more maintainable and could make it function as a plug-in. Introducing XML might influence the performance negatively and increase the overall size of the framework. An evaluation of these costs should be performed before implementing this functionality.

21.1.3 Optimalization

As mentioned above, the Bluetooth discovery times are currently too high for automatic exchange scenarios such as the the converging top ten list, etc. Several research projects around the world has looked at this. In [64], Woodings et.al. describe a strategy where they employ Infrared technology as support to speed up the connection time. In [59], Welsh et.al. discuss the use of Bluetooth in mobile environments such as vehicles. A study could be performed to look at ways to improve and work around the limitations in Bluetooth if this is not fixed in the next generation of Bluetooth devices.

In a mobile, battery powered device, computational operations drain the battery's power. In a Peer2Me piconet, the master node experiences a larger load than the slave nodes since e.g. all messages go through the master node. In order to level out all the nodes' battery life a load balancing mechanism should be introduced. This mechanism will include letting the master responsibility be shared among the nodes in the net over time. One load balancing scheme is described in [12].

21.2 Long Term Goals

In this section we will describe the long term goals of the Peer2Me framework.

21.2.1 Adopt New Technology

Currently only Bluetooth is supported as a network medium for Peer2Me but as more technologies are introduced on mobile phones an effort should be made to support these in Peer2Me. The current Bluetooth module should also be kept up to date as the Bluetooth specification evolves and is introduced in new versions on mobile phones.

The development of the JSR-259: Ad Hoc Networking API should be monitored closely to check how Peer2Me can be made to comply with the specification and or utilize the results from the JCP.

Thew JSR-271: MIDP 3 introduces a new set of features to the J2ME platform as specifies in Chapter 6. The Peer2Me framework should follow the development of MIDP 3 and be ported to it as soon as possible. Utilization of the new featrues in MIDP 3 increases the value of Peer2Me substancially.

21.2.2 Advanced Functionality

Automatic exchange applications continuously search for other devices to connect to. Searching for other devices takes up a lot of the network bandwidth, in addition it is also quite consumptional when it comes to battery power. In order to decrease the effort of searching, and adaptive discovery scheme can be employed. For instance, devices moving fast are more likely to encounter new nodes than devices being stationary. The JSR-179 specifies a Location API for J2ME that can provide a device's location to the application. By using this API, an auto discovery application can increase its search frequency based on how fast it is moving. An application may also maintain statistics over number of devices nearby to adjust its search frequency according to this.

The current mobile phones do not support scatternets at the hardware level, which limits the number of interconnected devices. When the mobile phones evolve further, scatternet implementations may be possible. Peer2Me today does not support links between every node in a network. By using scatternets, communication channels can be established between every node in the group, thus relieving the group's master of some of its routing duties. Taking scatternet implementation to the extreme and creating a full scale P2P net with Peer2Me is a quite large project in itself.

A study should be performed to check whether or not it is possible to implement support for the pure P2P model on mobile phones with Peer2Me. A pure P2P implementation using small scatternets would remove the need to have a master device routing all messages.

21.2.3 Empirical Work and Applications

To evaluate the value of mobile collaborative applications on mobile phones it should be conducted full scale end user testing. Applications that are stable and reliable should be spread around and adopted in a huge population. This way it should be possible to study how such applications affect the way humans collaborative using mobile devices. New kinds of applications and scenarios, that illustrates different kinds of usages of the framework, should be designed, implemented, tested and evaluated.

Part VII

Appendix

Appendix A

Discovery Time Statistics

This chapter contains empirical data gathered by a special MIDlet written to measure how many milliseconds it takes from a group search is initiated, to the device discovery completes and to the handshake protocol is complete. Tables A.1, A.2 and A.3 show the discovery times measured with each of our three test phones. All timestamps are measured in milliseconds. The first column shows the milliseconds spent from the start of the group search and to the completion of the device discovery. The second column shows the milliseconds spent from the start of the group search and to the handshake is finished and the two devices are joined in a group. All measurements are made in ideal environments, which means that the only active Bluetooth devices within communication range were the two mobile phones we used for the tests. When introducing more Bluetooth devices into the environment, the discovery times increase.

	Bluetooth discovery (ms)	Handshake protocol (ms)	Difference (ms)
Test 1	16546	19891	3345
Test 2	17375	20765	3390
Test 3	17203	20735	3532
Test 4	16359	19719	3360
Test 5	17265	20516	3251
Test 6	16619	19893	3274
Test 7	17118	20410	3292
Test 8	16405	20267	3862
Test 9	16922	20141	3219
Test 10	17085	20056	2971
Average	16889.7	20239.3	3349.6

Table A.1: Discovery times for Nokia 6600 as master and Sony Ericsson p900 as slave.

	Bluetooth discovery (ms)	Handshake protocol (ms)	Difference (ms)
Test 1	22297	25406	3109
Test 2	16438	19250	2812
Test 3	21015	24297	3282
Test 4	20015	23498	3483
Test 5	19203	21561	2358
Test 6	17266	20735	3469
Test 7	16531	19766	3235
Test 8	17437	20868	3431
Test 9	18797	21938	3141
Test 10	19265	22343	3078
Average	18826.4	21966.2	3139.8

Table A.2: Discovery times for Sony Ericsson p900 as master and Nokia 6600 as slave.

	Bluetooth discovery (ms)	Handshake protocol (ms)	Difference (ms)
Test 1	16080	20723	4643
Test 2	15110	18776	3666
Test 3	14963	18346	3383
Test 4	18300	23834	5534
Test 5	18281	21817	3536
Test 6	14986	18507	3521
Test 7	18222	24453	6231
Test 8	17502	22121	4619
Test 9	18254	23068	4814
Test 10	15753	20890	5137
Average	16745.1	21253.5	4508.4

Table A.3: Discovery times for Siemens s65 as master and Sony Ericsson p900 as slave.

Appendix B

The Peer2Me Developer Exercise

This chapter contains the exercise given to the participants of the developer testing workshop described in Chapter 17.

B.1 Introduction

This exercise gives developers practice in writing applications with the Peer2Me framework. At the end of this exercise you will have written a very simple chat application that can connect several devices in a network and allow each device's user to communicate.

For this exercise we have prepared an almost complete application which only misses the code for network communication and P2P infrastructure. The framework specific code will be added step by step. The code we will complete can be downloaded from the Peer2Me website [1].

This exercise assumes the reader has read and understood the Peer2Me tutorial.

B.2 Preparing the Framework

The first thing we'll have to do is to initialize the framework. In our example application the framework should be initialized after the user has pressed the "OK" button in the registration form. Create the necessary instance variables and instanciate and initialize the framework. This should be done at the start of the *ready* method. When preparing the framework, remember to create the service and register it with the framework. In addition to this, we'll set the framework's MessageSubscriber and GroupDiscoveryListener right away.

B.3 Searching for Other Devices

After the framework has been initialized, other devices running the same service has to be located. This can be done by starting a group search. Before starting the search, be sure to create a new

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

Group and set its service and add the group to the service. When a device is discovered, the slave device is notified through the *groupDiscovered* method. The slave device should automatically join the discovered group.

B.4 Monitoring the Group

Once a node has joined a group it should register a group monitor that can be notified of changes in the group, such as nodes leaving or joining, etc. This is done by implementing the methods defined by the *GroupMonitor* interface. For now, we will settle on implementing the *nodeJoined* method.

B.5 Sending a Message

The available code already has a text field where the user can enter messages that can be sent to the other nodes in the group. The next thing we have to do is get this text, wrap it in a message and send it.

B.6 Receiving a Message

Sending messages does not do much good if no one reads them. The next method to fix is the *messageReceived* method defined by the *MessageSubscriber* interface.

B.7 When Something Goes Wrong

Peer2Me is multi-threaded and in the Peer2Me class library we have defined an interface, *ExceptionHandler*, that can subscribe to exceptions from the framework threads. Modify the application so it implements the ExceptionHandler interface and displays the error messages on screen. The application also needs to register as an exception handler with the framework. This should be done together with the framework initialization.

Appendix C

Code Examples

This Chapter contains the complete source code for the two Peer2Me demonstration applications *TestSlave* and *TestMaster*. *TestSlave* is a simple MIDlet that listens for masters, offering a certain service. *TestMaster* searches for available slave devices and connects to each found slave device. *TestSlave*, upon receiving a notification of a new Node joining its group, sends a simple message to the joining node. Listings C.1 and C.2 show the source code for *TestSlave* and *TestMaster* respectively.

Listing C.1: Complete source code for TestSlave

```
1  package no.ntnu.idi.mowahs.applications.newtestapps;
2
3  import java.util.Enumeration;
4
5  import javax.microedition.lcdui.Display;
6  import javax.microedition.lcdui.Form;
7  import javax.microedition.midlet.MIDlet;
8  import javax.microedition.midlet.MIDletStateChangeException;
9
10 import no.ntnu.idi.mowahs.project.domain.Group;
11 import no.ntnu.idi.mowahs.project.domain.Message;
12 import no.ntnu.idi.mowahs.project.domain.Node;
13 import no.ntnu.idi.mowahs.project.domain.Service;
14 import no.ntnu.idi.mowahs.project.domain.TextMessagePart;
15 import no.ntnu.idi.mowahs.project.framework.ExceptionHandler;
16 import no.ntnu.idi.mowahs.project.framework.Framework;
17 import no.ntnu.idi.mowahs.project.framework.GroupDiscoveryListener;
18 import no.ntnu.idi.mowahs.project.framework.GroupMonitor;
19 import no.ntnu.idi.mowahs.project.framework.MessageSubscriber;
20
21 public class TestSlave extends MIDlet implements GroupDiscoveryListener,
22         ExceptionHandler, MessageSubscriber, GroupMonitor {
23
24     public static final String SERVICE_ID = "TESTAPP";
25
26     private Form form;
27     private Display display;
28     private Framework myFramework;
29     private Service myService;
30
```

```
31      private void writeText(String text){
32          form.append(text + '\n');
33          display.setCurrent(form);
34      }
35
36      protected void startApp() throws MIDletStateChangeException {
37          display = Display.getDisplay(this);
38          form = new Form("TestSlave");
39          myService = new Service(SERVICE_ID);
40          writeText("Startet testSlave...");
41          myFramework = Framework.getInstance("slavetester", "testing slave", "no
                .ntnu.idi.mowahs.project.bluetooth.network.BluetoothNetwork");
42          myFramework.init();
43          myFramework.registerService(myService);
44          myFramework.setMessageSubscriber(this);
45          myFramework.setExceptionHandler(this);
46          myFramework.setGroupDiscoveryListener(this);
47
48      }
49
50      protected void pauseApp() {
51
52      }
53
54      protected void destroyApp(boolean arg0) throws MIDletStateChangeException {
55
56      }
57
58
59
60
61      public void groupDiscovered(Group group) {
62          group.setMonitor(this);
63
64          writeText("Received group!");
65          writeText("Master:" + group.getMaster().getKey());
66          Enumeration enum = group.getSlaves().elements();
67          while(enum.hasMoreElements()){
68              writeText("Slave: " + ((Node)enum.nextElement()).getKey());
69          }
70
71          myFramework.joinGroup(group);
72
73      }
74
75      public void handleException(Exception e) {
76          writeText(e.getMessage());
77
78      }
79
80      public void messageReceived(Message message) {
81          writeText(message.getMessagePart("message").getFieldValue());
82
83      }
84
85      public boolean runningService(String serviceID) {
86          if(serviceID.equals(myService.getServiceID()))return true;
```

```
87          return false;
88      }
89
90      public void exit() {
91          notifyDestroyed();
92      }
93
94      public void allowJoin(Group group, Node node) {
95
96      }
97
98      public void nodeJoined(Group group, Node node) {
99          writeText("Node joined: " + node.getKey());
100
101         Message message = new Message();
102         TextMessagePart text = new TextMessagePart();
103         text.setDescription("message");
104         text.setFieldValue("hello slave");
105         message.addMessageBodyPart(text);
106         message.addRecipient(node);
107
108         myFramework.sendMessage(message, myService);
109
110
111     }
112
113     public void nodeLeft(Group group, Node node) {
114         writeText("Node left: " + node.getKey());
115
116     }
117
118 }
```

Listing C.2: Complete source code for TestMaster

```
1  package no.ntnu.idi.mowahs.applications.newtestapps;
2
3  import javax.microedition.lcdui.Display;
4  import javax.microedition.lcdui.Form;
5  import javax.microedition.midlet.MIDlet;
6  import javax.microedition.midlet.MIDletStateChangeException;
7
8  import no.ntnu.idi.mowahs.project.domain.Group;
9  import no.ntnu.idi.mowahs.project.domain.Message;
10 import no.ntnu.idi.mowahs.project.domain.Node;
11 import no.ntnu.idi.mowahs.project.domain.Service;
12 import no.ntnu.idi.mowahs.project.framework.ExceptionHandler;
13 import no.ntnu.idi.mowahs.project.framework.Framework;
14 import no.ntnu.idi.mowahs.project.framework.GroupMonitor;
15 import no.ntnu.idi.mowahs.project.framework.MessageSubscriber;
16
17 public class TestMaster extends MIDlet implements ExceptionHandler,
18         MessageSubscriber, GroupMonitor {
19     public static final String SERVICE_ID = "TESTAPP";
20
21     private Form form;
22     private Display display;
```

```
23      private Framework myFramework;
24      private Service myService;
25      private Group myGroup;
26
27      private void writeText(String text){
28          form.append(text + '\n');
29          display.setCurrent(form);
30      }
31
32      protected void startApp() throws MIDletStateChangeException {
33          display = Display.getDisplay(this);
34          form = new Form("TestMaster");
35          myService = new Service(SERVICE_ID);
36          myGroup = new Group();
37
38          writeText("Startet testMaster...");
39          myFramework = Framework.getInstance("mastertester", "testing master", "
                no.ntnu.idi.mowahs.project.bluetooth.network.BluetoothNetwork");
40          myFramework.init();
41          myFramework.setMessageSubscriber(this);
42          myFramework.setExceptionHandler(this);
43
44          myGroup.setMaster(myFramework.getLocalNode());
45          myGroup.setMonitor(this);
46          myService.setGroup(myGroup);
47
48          myFramework.registerService(myService);
49
50          myFramework.startGroupSearch(myService);
51
52
53      }
54
55      protected void pauseApp() {
56      }
57
58      protected void destroyApp(boolean arg0) throws MIDletStateChangeException {
59      }
60
61      public void handleException(Exception e) {
62          writeText(e.getMessage());
63
64      }
65
66      public void messageReceived(Message message) {
67      }
68
69      public boolean runningService(String serviceID) {
70          return false;
71      }
72
73      public void exit() {
74      }
75
76      public void allowJoin(Group group, Node node) {
77      }
78
```

```
79      public void nodeJoined(Group group, Node node) {
80          writeText("Node joined: " + node.getKey());
81
82      }
83
84      public void nodeLeft(Group group, Node node) {
85          writeText("Node left: " + node.getKey());
86
87      }
88
89  }
```

Appendix D

Questionnaire for Peer2Me developer testing

D.1 Background and Experiences

1. For how many years have you been developing software?
2. Describe your programming skills.

   - Professional
   - Good
   - Medium
   - Poor
   - None

3. For how many years have you been developing software with Java?
4. Describe your Java skills.

   - Professional
   - Good
   - Medium
   - Poor
   - None

5. Have you ever developed a mobile application before?

   - Yes

- No

6. Have you ever developed using J2ME?

- Yes
- No

7. Have you ever developed a Bluetooth application?

- Yes
- No

8. Do own a mobile phone supporting Bluetooth?

- Yes
- No

D.2 The Domain Concepts

9. I think the domain concepts of Peer2Me are easy to understand.

- Completely agree
- Agree
- Neutral
- Disagree
- Completely disagree

10. I think the concept Framework is easy to understand.

- Completely agree
- Agree
- Neutral
- Disagree
- Completely disagree

11. I think the concept Node is easy to understand.

- Completely agree
- Agree
- Neutral
- Disagree
- Completely disagree

12. I think the concept Network is easy to understand.

- Completely agree
- Agree
- Neutral
- Disagree
- Completely disagree

13. I think the concept Service is easy to understand.

- Completely agree
- Agree
- Neutral
- Disagree
- Completely disagree

14. I think the concept Group is easy to understand.

- Completely agree
- Agree
- Neutral
- Disagree
- Completely disagree

15. I think the concept Message is easy to understand.

- Completely agree
- Agree

- Neutral
- Disagree
- Completely disagree

16. I think these concepts simplifies the problem domain.

- Completely agree
- Agree
- Neutral
- Disagree
- Completely disagree

D.3 The Peer2Me Development Guide

17. I think the development guide is easy to read.

- Completely agree
- Agree
- Neutral
- Disagree
- Completely disagree

18. I think the development guide helped me through the exercise.

- Completely agree
- Agree
- Neutral
- Disagree
- Completely disagree

19. Do you have any suggestions as to how the guide could be made more readable?
20. Do you have any suggestions as to how the guide could be made more useful?

D.4 The Exercise

21. What do you think was the most difficult part of the exercise?

- Preparing the Framework
- Searching for Other Devices
- Monitoring the Group
- Sending a Message
- Receiving a Message
- When Something Goes Wrong

22. What was the most time consuming part of the exercise for you?

- Preparing the Framework
- Searching for Other Devices
- Monitoring the Group
- Sending a Message
- Receiving a Message
- When Something Goes Wrong

23. What were the problems that you experienced, if any?

D.5 Summary

24. I think Peer2Me speeds up the development of collaborative mobile applications.

- Completely agree
- Agree
- Neutral
- Disagree
- Completely disagree

25. I think Peer2Me clarifies the domain of mobile collaborative applications.

- Completely agree

- Agree
- Neutral
- Disagree
- Completely disagree

26. Do you have any other comments, criticisms or suggestions?

Appendix E

Dictionary

API Application Programming Interface. An API is a set of definitions of the ways one piece of computer software communicates with another. It is a method of achieving abstraction, usually (but not necessarily) between lower-level and higher-level software. One of the primary purposes of an API is to provide a set of commonly-used functions that can be used by a programmer to design software.

Bluetooth An open standard for wireless transmission of voice and data between mobile devices in a PAN.

CDC The Connected Device Configuration. A framework for building J2ME applications on all kind of embedded devices ranging from a pager up to Set-top box. The CLDC and MIDP specifications allow a more fine-grained functionality for more specialized devices.

CLDC Connected, Limited Device Configuration. A specification that defines a subset of classes defined in the J2SE and the CDC. The classes provide basic functionality that can be utilized on a mobile device.

CSCW Computer Supported Cooperative Work. The CSCW field deals with the use of computers to support cooperation between people.

Framework An object oriented framework can be defined as a set of classes that embodies an abstract design for solutions to a family of related problems. Frameworks are often designed to simplify a problem domain and open for reuse and thereby decrease the development time and effort.

IDI Department of Computer and Information Science

IEEE Institute of Electrical and Electronics Engineers. An organization providing standards to the technical society.

GQM Goal Question Metric. The GQM method is a formal method to break down goals of an empirical study or an experiment to a set of questions and related measurement metrics. The GQM method forces scientists to decide upon and define what they actually want to measure before doing it.

J2ME Java 2 Micro Edition. J2ME is a development platform owned and developed by Sun Microsystems aimed at small and limited devices. Today most mobile phones has support for running J2ME applications. J2ME consists of two programming specifications, CLDC and MIDP.

J2SE Java 2 Platform, Standard Edition. Provides a complete environment for applications development on desktops and servers and for deployment in embedded environments.

JABWT Java APIs for Bluetooth Wireless Technology. JABWT are a set of standard Java APIs that enable the development of applications in Java conforming to the Bluetooth Specification 1.1. JABWT is an implementation of the JSR 82 specification. The JSR-82 Expert Group was responsible for defining the standard.

JSR-82 Java Specification Request 82. The specification standardizes a set of Java APIs to allow Java-enabled devices to integrate into a Bluetooth environment. JABWT implement this specification.

MANET Mobile Ad Hoc NETworks. Mobile ad hoc network networks that are spontaneous, self-configuring and wireless with no fixed infrastructure. A mobile ad hoc network consists of mobile nodes that use a wireless interface to send packet data.

MIDlet A J2ME application that conforms with CLDC and MIDP and is intended for mobile devices. All Peer2Me applications are implemented as MIDlets.

MIDP Mobile Information Device Profile. MIDP sits on top of the CLDC and provides additional functionality to a J2ME application related to the user interface, networking, and messaging.

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

Multihop network A multihop network is a network where some nodes are out of reach from each other and cannot communicate directly. Network traffic between nodes may have to be forwarded by other intermediate nodes.

NTNU The Norwegian University of Science and Technology.

P2P Peer-to-Peer. Peer-to-peer computing refers to a class of applications that enables users to form logical networks on top of any infrastructure and to share and exchange digital content. Nodes in a P2P network, called peers, function as both server and clients.

PAN Personal Area Network. A PAN is one class of MANETs A PAN is mainly used for connecting devices within a limited area with a radius of a few meters. Example devices forming a PAN are laptop computers, printers and mobile phones and various peripheral devices such as keyboards and mice. The communication range of a PAN is normally up to 10 meters.

Peer2Me Peer-to-peer for J2ME. A framework for mobile collaboration on mobile phones utilizing J2ME and Bluetooth.

Piconet A PAN can either be a piconet or a scatternet. In a piconet is all devices in the network interconnected in a singlehop network. A Bluetooth piconet has a single master device and up to seven slave devices.

RFCOMM RS232 Serial Cable Emulation Profile. A Bluetooth transport protocol in the core protocol stack simulating a serial port connection

Scatternet A PAN can either be a piconet or a scatternet. In a scatternet nodes may be out of reach from each other an have to communicate in a multihop fashion.

Singlehop network A singlehop network is a network where nodes can communicate directly with each other. All nodes in the network are in reach of each other.

USB Universal Serial Bus. A wired protocol for interconnecting devices in a PAN.

Bibliography

[1] Mobile ad-hoc netze. http://www-i4.informatik.rwth-aachen.de/~mesut/manet/display.html, 2002. (Accessed: 15.09.2004).

[2] Rockyroad white paper. http://www.jrra.org/pub/WhitePaper.zip, 2002. (Accessed: 12.04.2005).

[3] Getting started with bluetooth. http://developers.sun.com/techtopics/mobility/midp/articles/bluetooth1/, 2004. (Accessed: 25.08.2004).

[4] Jsr 259: Ad hoc networking api. http://www.jcp.org/en/jsr/detail?id=259, 2004. (Accessed: 02.20.2005).

[5] Rockyroad/jrra. http://www.jrra.org, 2004. (Accessed: 12.04.2005).

[6] Freescale demos uwb mobile. http://www.electronicsweekly.com/Article38447.htm, 2005. (Accessed: 02.02.2005).

[7] Model view controller. http://en.wikipedia.org/wiki/Model-view-controller, 2005. (Accessed: 13.05.2005).

[8] Technology news from philips. http://www.semiconductors.philips.com/news/content/file_1146.html, 2005. (Accessed: 02.05.2005).

[9] Telecom products outlook 2005. http://www.telecom.globalsources.com/gsol/I/Bluetooth-handsfree/a/9000000057461.htm, 2005. (Accessed: 02.05.2005).

[10] T. . T. Association. *Protocol Adaption Layer (PAL) for IEEE 1394 over IEEE 802.15.3*, 2004.

[11] V. R. Basili. The Experimental Paradigm in Software Engineering. In H. D. Rombach, V. R. Basili, and R. W. Selby, editors, *Experimental Software Engineering Issues: Critical Assessment and Future Directions*, pages 3–12, Dagstuhl Castle, Germany, September 14-18 1992. Springer Verlag. LNCS 706.

[12] M. Boulkenafed, D. Sacchetti, and V. Issarny. Using group management to tame mobile ad hoc networks. In E. Lawrence, B. Pernici, and J. Krogstie, editors, *Mobile Information Systems*, pages 245–261. Springer, 2004.

[13] S. S. Bygdås, Øystein Myhre, S. Nyhus, T. Urnes, and Åsmund Weltzien. Bubbles: Navigating content in mobile ad-hoc networks. Technical report, Telenor FOU, 2003.

[14] Club-java.com. Midp java phone benchmark. http://www.club-java.com/TastePhone/J2ME/MIDP_Benchmark.jsp. (Acessed: 1.6.2005.

[15] M. Conti. Body, personal, and local ad hoc wireless networks. Technical report, Consiglio Nazionale delle Ricerche, 2003.

[16] M. Conti. Peer-to-peer research at stanford. Technical report, Computer Science Department, 2003.

[17] T. B. Corporation. Bedd. http://www.bedd.com. (Accessed: 12.04.2005).

[18] T. Dingsøyr, T. Stålhane, and N. B. More. Lightweight post mortem reviews. Technical report, Sintef Telecom and Informatics, 2002.

[19] Dyba, Wedde, Stalhane, B. Moe, Conradi, Dingsoyr, Sjoberg, and Jorgensen. Spiq, software process improvement for better quality. Technical report, Ntnu and Sintef, 2000.

[20] G. T. Edwards, D. C. Schmidt, and A. Gokhale. Integrating publisher/subscriber services in component middleware for distributed real-time and embedded systems. In *ACM-SE 42: Proceedings of the 42nd annual Southeast regional conference*, pages 171–176, New York, NY, USA, 2004. ACM Press.

[21] C. A. Ellis, S. J. Gibbs, and G. Rein. Groupware: some issues and experiences. *Commun. ACM*, 34(1):39–58, 1991.

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

[23] E. Gamme, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.

[24] H. Hedlund. Bleutooth baseband specification, version 1.1. www.bluetooth.com.

[25] A. Heinemann, J. Kangasharju, F. Lyardet, and M. Mühlhäuser. Ad hoc collaboration and information services using information clouds. In T. Braun, N. Golmie, and J. Schiller, editors, *Proceedings of the 3rd Workshop on Applications and Services in Wireless Networks, (ASWN 2003)*, pages 233–242, Bern, Switzerland, 2003. Institute of Computer Science and Applied Mathematics, University of Bern.

[26] A. Heinemann, J. Kangasharju, F. Lyardet, and M. Mühlhäuser. iclouds – peer-to-peer information sharing in mobile environments. In H. Kosch, L. Böszörményi, and H. Hellwagner, editors, *Euro-Par 2003. Parallel Processing, 9th International Euro-Par Conference*, volume 2790 of *Lecture Notes in Computer Science*, pages 1038–1045, Klagenfurt, Austria, 2003. Springer.

[27] L. Holmquist, J. Falk, and J. Wigstrm. Supporting group collaboration with inter-personal awareness devices, 1999.

[28] B. Hopkins and R. Antony. *Bluetooth for java*. 2003.

[29] B. Hui. Go wild wirelessly with bluetooth and java. http://jdj.sys-con.com/read/43551.htm, 2004. Accessed: 16.05.2005.

[30] IEEE. *IEEE Standard for Information technology telecommunications and information exchange between system - Local and metropolitan area networks Specific Requirements Part 15.3: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for High Rate Personal Area Networks (WPANs)*, 2003.

[31] R. E. Johnson and B. Foote. Designing reusable classes. *Journal of Object-Oriented Programming*, 1(2):22–35, 1988.

[32] L. Kirkhus and A. R. Sveen. An examination of mobile devices for spontaneous collaboration. Technical report, Institutt for datateknikk og informasjonsvitenskap, 2003.

[33] L. Kirkhus and A. R. Sveen. Mowahs - mobile collaboration framework. Technical report, Institutt for datateknikk og informasjonsvitenskap, 2004.

[34] G. Kortuem. Proem: A peer-to-peer computing platform for mobile ad hoc networks. Technical report, Wearable Computing Laboratory, Department of Computer Science, 2001.

[35] G. Kortuem. A methodology and software platform for building wearable communities. Technical report, University of Oregon, 2002.

[36] G. Kortuem, J. Schneider, 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 mobiel ad hoc networks. Technical report, 2001.

[37] C.-H. W. Lund and M. S. Norum. A framework for mobile collaborative applications on mobile phones. Technical report, Department of Computer and Information Science, 2004.

[38] P. J. Magnus Frodigh and P. Larsson. Wireless ad hoc networking - the art of networking without a network. Technical report, Ericsson, 2000.

[39] S. Microsystems. Java 2 micro edition. `http://java.sun.com/j2me/docs/j2me-ds.pdf`, 2002. (Accessed: 21.5.2005).

[40] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. Technical report, Hewlett-Packard Company,HP Laboratories Palo Alto, 2002.

[41] NASA. Goal question metric. `http://sel.gsfc.nasa.gov/website/exp-factory/gqm.htm`. (Accessed: 27.04.2005).

[42] Nokia. Series 80 developer platform: Ad-hoc communications over wlan. Technical report, Nokia, 2005.

[43] P. og Teletilsynet. Det norske telemarkedet 1. halvår 2004.

[44] J. S. Olson, S. Teasley, L. Covi, and G. Olson. The (currently) unique advantages of collocated work. chapter 5, pages 113–135. MIT Press, 2002.

[45] C. E. Ortiz. Using the java apis for bluetooth wireless technology. Technical report, Sun Microsystems, 2004.

[46] J. C. Process. Jsr 271: Mobile information device profile 3. http://www.jcp.org/en/jsr/detail?id=271, 2005. Accessed: 13.05.2005.

[47] T. Reenskaug. Thing-model-view-editor. Technical report, Institutt for informatikk, University of Oslo, 1979.

[48] D. Schoder and K. Fischbach. Peer-to-peer, anwendungsbereiche und herausforderungen, in: Schoder detlef; fischbach, kai; teichmann, rene: Peer-to- peer (p2p), okonomische, technologische und juristische perspektiven. Technical report, 2002.

[49] SMlab. Gqm applications method. http://irb.cs.uni-magdeburg.de/sw-eng/us/java/GQM/. (Accessed: 27.04.2005).

[50] R. Software. Rococo training: Bluetooth seminar for technologists. http://www.rococosoft.com/docs/bluetooth_seminar.pdf, 2005. Accessed: 15.05.2005.

[51] R. Software. Rococo training: Building bluetooth applications in java. http://www.rococosoft.com/docs/java_apps.pdf, 2005. Accessed: 15.05.2005.

[52] R. Software. Rococo training: Standard java apis for bluetooth. http://www.rococosoft.com/docs/java_apis.pdf, 2005. Accessed: 15.05.2005.

[53] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa a wide-area distributed database system. Technical report, Department of Electrical Engineering and Computer Sciences, 1996.

[54] A. Sutcliffe. Scenario-based requirements analysis. *Requirements Engineering*, 3(1):48–65, 1998.

[55] Teleca. Teleca bluetooth academy website. http://www.comtec.teleca.se/info_academy2.asp, 2005. Accessed: 15.05.2005.

[56] F. Tétard, E. Patokorpi, and V. Kadytė. User-centered design of mobile services for tourists. In *Mobile Information Systems*, pages 155–168. Springer, 2004.

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

[58] I. TranSafety. Researchers study the walking speeds of older pedestrians. http://www.usroads.com/journals/rej/9704/re970404.htm, 1997. Accessed: 24.5.2005.

[59] E. Welsh, P. Murphy, and P. Frantz. Improving Connection Times for Bluetooth Devices in Mobile Environments. In *International Conference on Fundamentals of Electronics Communications and Computer Sciences (ICFS)*, March 2002.

[60] M. Wiberg and Åke Grønlund. Exploring mobile cscw: Five areas of questions for further research.

[61] C. Wille, N. Brehmer, and R. R. Dumke. Software measurement of agent-based systems. Technical report, Otto-von-Guericke-Universität Magdeburg. Institut für Verteilte Systeme, 2004.

[62] B. J. Wilson. *JXTA, second edition*. 2002.

[63] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslen. *Experimentation in software engineering*. Kluwer Academic Publishers, 2000.

[64] R. Woodings, D. Joos, T. Clifton, and C. D. Knutson. Rapid heterogeneous connection establishment: Accelerating bluetooth inquiry using irda.