

# A PEER-TO-PEER FRAMEWORK FOR MOBILE COLLABORATION

Alf Inge Wang  
Dept. of Computer and Information Science  
Norwegian University of Science and Technology  
N-7491 Trondheim, Norway  
alfw@idi.ntnu.no

Michael Sars Norum  
BearingPoint Norway AS  
N-0051 Oslo, Norway  
michael.norum@bearingpoint.com

Carl-Henrik Wolf Lund  
Bekk Consulting AS  
N-0102 Oslo, Norway  
Carl-Henrik.Lund@bekk.no

## ABSTRACT

The paper describes a framework used to develop mobile collaborative applications in Java 2 Micro Edition (J2ME). The framework focuses on the support for same-place-same-time collaboration on mobile phones utilising personal area networks (PANs). We believe that the PAN technology available on mobile phones enables for a broad range of new collaborative applications supporting collocated work and spontaneous interaction. Today, such applications can be developed using J2ME along with PAN APIs like the Bluetooth API supported by the most recent mobile phones. However, these APIs lack of high-level peer-to-peer primitives, so the developers have to focus too much on the technical aspects and not the collaborative aspects. Our framework provides the necessary high-level concepts to develop collaborative peer-to-peer applications and is independent of the underlying PAN technology. This paper describes the concepts, the design, and results from developing applications using our framework.

## KEY WORDS

Software Engineering Applications, Mobile and Wireless Computing, Peer-to-peer, J2ME, Bluetooth.

## 1 Introduction

Over the last decade, the mobile phone has become so common and so much used that it can be seen as a natural extension of the human body. In some countries like in Scandinavia, more than 90% of the population own and use mobile phones. The revolution in mobile hardware has changed mobile phones from being "just phones" to powerful versatile computers communicating through cellular networks and personal area networks (PANs). 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 technologies enable devices to detect and connect to other devices that are in sufficient proximity, and form mobile peer-to-peer (P2P) networks. P2P networks are 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, mobile phones can improve face-to-face interaction. Technologies like Bluetooth (BT) and Infrared (IR) are well supported on mobile phones. Utilising PAN technology enables for a broad range of new categories of collaborative applications supporting collocated work and spontaneous interaction. Different kinds of application scenarios have been described by Kortuem in [5] and by Heinemann in [10].

Most work in the research area of mobile collaboration has resulted in specialised prototypes of mobile P2P and collaborative applications to demonstrate the potential of the emerging technologies. Our focus has been on providing a framework for rapid development of mobile collaborative applications for mobile phones utilising the existing support for J2ME and PAN. The architecture and concepts of our framework are independent of the kind of PAN technology supported in the mobile device.

## 2 Mobile Computer Supported Cooperative Work

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 [8], several advantages of collocating a work force are 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. 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 [2], two dimensions, time and place comprise the domain of CSCW, where time is divided into *Real time* and *Asynchronous* and place is divided into *Same place* and *Different place*. 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, make 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. Our framework covers both real time and asynchronous applications in the "Same Place" category in a mobile environment.

The targeted platform for our framework is mobile phones or similar mobile devices. Use mobile phones as an execution platform for CSCW applications has several advantages over traditional CSCW. Firstly, since the mobile phones are personal so they can be used to identify a user. Secondly, 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. Thirdly, mobile phones can almost be considered to be always on, always present. Due to this, someone using his or her mobile phone for CSCW purposes will achieve a high degree of user availability.

Modern mobile phones support more than one communication 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 in a decentralized manner. These characteristics relate strongly to the nature of human spontaneity, which make PANs suitable for making spontaneous collaborative applications. A PAN creates a digital sphere around a person. The communication range of the PAN limits this digital sphere.

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. 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). Impromptu collaboration can involve different degrees of user interaction:

**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 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). Our framework focuses mainly on support for impromptu collaborative applications.

### 3 A Mobile Collaboration Framework

This section presents the concepts, the architecture, the design and the protocols of our framework.

#### 3.1 The 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 was to make the framework independent of the underlying network technology. This will reduce the effort to migrate the framework to a new network medium. Figure 1 shows our semi-layered architecture for the framework.

The leftmost part in the figure shows the layered view of the framework. Applications use the interface provided by the framework's core functionality. The Framework uses a generic interface to control technology specific Network modules. The framework entity is the core entity and is used as an interface between the application and the rest of system. It manages resources such as known peers and available network mediums. The technology specific network module can be implemented with technologies such as BT, ZigBee or WLAN. The module and Network interface layers are what makes the framework independent of

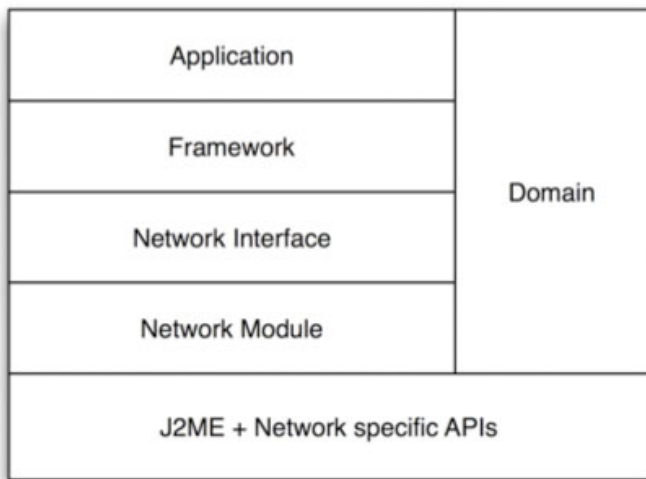


Figure 1. The High-level Architecture

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. A *node* is a logical representation of a peer, in our case a mobile phone running the framework. A *service* is connected to a specific application and supported by zero or more nodes. The service is used to identify a part of the application shared by several users. 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. 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.

Of the concepts described above, two are dependent on network technology: The network and node entities. For those two, the framework uses abstractions 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.

### 3.2 The Framework Design

The framework consists of five packages: Domain, Framework, Network, BT and Util.

The **Domain** package consists of the following classes for maintaining a view of the runtime environment of the applications: *service*, *node*, *group* and *message*. Note since the communication in the framework uses a master-slave protocol, there exist two types of nodes - one for master and one for slave.

The **Framework** package contains the application's

main interface for controlling the functionality of the framework. The framework is instantiated, 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. The package has also classes for monitoring group activities, subscription to messages and an exception handler.

The **Network** package provides the services and primitives required by the framework to run independently of a network implementation. The Network package contains only the *Network* class. This class is an abstract class providing 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. The current implementation of the framework only includes a BT network module. This module can be used as a starting-point for how to implement for other network transport mediums.

The **BT** package contains the BT implementations of the general domain and network packages. This package contains BT versions of the *node* and the *network* classes. Further, this package contains classes to handle discovery of mobile devices running the same service and to manage connection between devices. Most of the effort of the project was put into implementation of this package.

Finally, the **Util** package provides support functionality for the applications using the framework. Currently the package only contains classes to manage persistent storage of data.

### 3.3 The Protocols used

The three protocols described in this section provide basic functionality. The protocols have evolved throughout the whole project to become mature, fault-tolerant and complete.

#### 3.3.1 The Handshake Protocol

When devices connect as nodes in a group, certain messages need to be exchanged as a handshake. The handshake is carried out with 4 messages pr. node and complemented with up to 5 messages to existing members of the group. Figure 2 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.

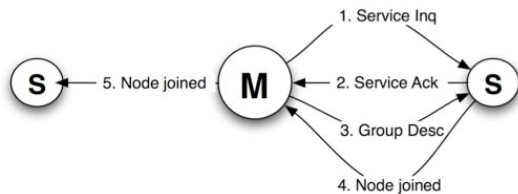


Figure 2. Messages in the handshake protocol

### 3.3.2 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.

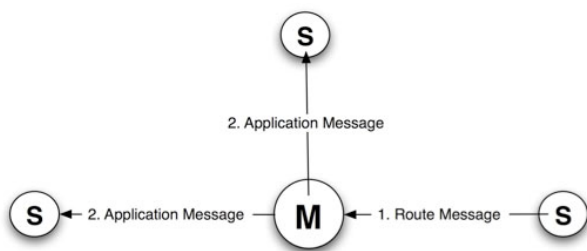


Figure 3. Messages in the routing protocol

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 3 shows a graphical representation of the route protocol.

### 3.3.3 The Disconnection Protocol

The framework detects when a connection to another node goes down, 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 below.

```

For each grp registered on the local node:
  If the local node is the master of the grp:
    If the disconnect node was a slave in that grp:
      Remove the node from the grp
      Send node left message to all other nodes in grp
  If the local node is not the master:
    If the disconnect node is the master of grp:
      Remove the grp from the local node
    
```

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.

## 4 Developing Mobile Collaborative Applications

We will here illustrate the initialisation of a master in our framework.

A mobile collaborative application using our framework must be implemented as a MIDlet in J2ME. A MIDlet is initiated by calling the method `startApp()`. The listing on next page shows an example of how our framework is initiated in `startApp`. The first 7 lines are just to create a GUI in J2ME (line 2 and 3) and declaration of variables. Before the framework can be used, it must be initialised by setting certain properties (line 7). The Framework instance is first retrieved by feeding the node name, the node description and the network name to the `getInstance` method. It is then initialized by the `init` method. The `init` method prepares the framework for use with the arguments given previously. Line 9 initiates the message-handling while line 10 enables exception handling. Line 12 is used to put current device as a master device. Line 13 is used for initiating group monitoring for managing nodes that do not respond and new nodes. Line 14 and 15 relate the group to the service and vice versa. In line 16, the current device registers himself as a slave (to get the same services that all other devices). Finally, line 18 initiates a search for devices in the area.

To initial a slave device, the lines 5, 12, 13, 14 and 15 can be skipped. In addition, the slave must add a line to make it possible for the master to discover the device.

```

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      my Framework = Framework.getInstance("mastertester", "testingmaster",
8          "no.ntnu.idi.mowahs.project.bluetooth.network.BTNetwork");
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 }

```

To send messages in our framework, the messages are sent to the service. All the devices that have initiated the *MessageSubscriber* interface will receive messages.

## 5 Experiences from using Our Framework

To test our framework, we have developed three prototype applications using the framework: A *Business Card Exchange* application where business cards can be exchanged between people with mobile phones that are collocated, a *PAN Instant Messaging (IM)* application, and a *Converging Top Ten List* application (e.g. collect the top-ten best beer prices at a campus area). These applications were tested both on simulators and in a real environment using three students equipped with three different mobile phones - a Sony Ericsson P900, a Nokia 6600 and a Siemens S65. At the time we performed the tests, these were the only mobile phones available with support for BT in J2ME. The tests of the three applications were all successful, although we discovered a serious limitation in the BT API in J2ME. The problem was that the J2ME application was halted during a BT device discovery process that caused the users to wait up to 20 seconds before the devices could start to interact. If new devices were to be added or removed from the group, the same process had to be performed. The users of the tests were satisfied with the functionality of the applications, but the long delays of discovery ruined the usefulness of the applications.

In addition to test the functionality of the framework, we also tested how hard the framework was to use for developers without experience from using the framework. To do this, we arranged a workshop where 7 last year master students participated. The workshop was divided into three parts: Education, development and evaluation. In the *education session*, the participants got a 20-minute lecture describing the purpose of the workshop and giving an introduction to the framework. In the *development session*, we gave every participant the source code of a simple chat application missing code for management of network communication and P2P infrastructure. The task was to finish

the application to work by adding network and P2P code by using the framework. During the task, the participants measured the time they spent on training (lecture + reading documentation) and time spent on programming. In the *evaluation session*, we performed a post-mortem analysis of the workshop and summarised the data. The data showed that 5 out of 7 managed to participate to participants managed to complete the task, while the 2 remaining completed 90 % of the task. The average time spent on training was 40 minutes and the average time spent on programming was 86 minutes. This result was compared to an estimate of the time for developing a similar application using the BT API directly in J2ME. These times, based on experiences from several BT developers, were estimated to be to be 8 hours on training and 8 hours on development. Also we compared the lines of code needed to be written for a simple chat application using our framework (108 lines) and reference implementation (BlueChat by Ben Hiu) without the framework (586 lines). The simple chat application was a bit simpler than the BlueChat, but even for a more complex application (the PAN Instant Messaging) the lines of code were 242. The data clearly shows that our framework reduces the development and training effort drastically for peer-to-peer applications in J2ME.

## 6 Related Work

BEDD is a software suite, developed and maintained by the BEDD Corporation [1]. 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 smart phones. BEDD is a commercial project and not a research or open source project. 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 BT as a transport medium for exchanging messages between devices that are running the BEDD software and that are within BT range of each other.

The JSR-259 Ad Hoc Networking API is a Java Com-

munity 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. No implementation is available yet.

RockyRoad is a P2P protocol for both J2ME and J2SE [9]. The RockyRoad implementation is directed at wireless networking technologies and currently has 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.

iClouds is an architecture for impromptu collaboration and spontaneous exchange of information between users that come within each other's digital sphere [3]. 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.

JXTA is a Java framework for creating P2P networks. Two different projects exist for porting JXTA to wireless mobile devices, JXME proxied and JXME proxyless [7]. 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 are available with support for PAN, only TCP/IP.

PROEM is an open computing platform that provides a complete solution for developing and deploying P2P applications for MANETs [6]. The platform is based on experiences from developing a series of mobile applications. PROEM is implemented as a platform independent framework using J2SE. Today's mobile devices only support J2ME and can therefore not use the PROEM platform for developing P2P applications.

Compared to the projects described above, our project has the only working peer-to-peer framework for J2ME that runs on mobile phones.

## 7 Conclusion

Our framework was indented to provide support for rapid development mobile collaborative applications. Our preliminary tests show that we have succeeded in this respect. We are currently developing new applications to further test the framework and we will continue to improve the framework based on these tests. The current version of the framework consists of 1163 lines of code spread out in 26 classes. The size of a deployable jar-file is about 11KB, which means that the framework is well suited for devices

with limited memory and CPU-power.

## Acknowledgement

We would like to thank Lars Kirkhus and Anders R. Sveen for their contribution to the framework.

## References

- [1] T. B. Corporation. Bedd community. <http://www.bedd.com>, 2006.
- [2] Clarence A. Ellis, Simon J. Gibbs, and Gail Rein. Groupware: some issues and experiences. *Commun. ACM*, 34(1):39–58, 1991.
- [3] Andreas Heinemann, Jussi Kangasharju, Fernando Lyardet, and Max Mühlhäuser. iClouds – Peer-to-Peer Information Sharing in Mobile Environments. In Harald Kosch, László Böszörményi, and Hermann 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.
- [4] jcp.org. Jsp 259: Ad hoc networking api. <http://www.jcp.org/en/jsr/detail?id=259>, 2006.
- [5] Gerd Kortuem, Jay Schneider, Dustin Preuit, Thaddeus G. C. Thompson, Stephen Fickas, and Zary Segall. When Peer-to-Peer comes Face-to-Face: Collaborative Peer-to-Peer Computing in Mobile Ad hoc Networks. In *First International Conference on Peer-to-Peer Computing (P2P'01)*, pages 75–94, Washington, DC, USA, 2001. IEEE Computer Society.
- [6] Gerd Kortuem, Jay Schneider, Dustin Preuit, Thaddeus, G. C. Thompson, Stephen Fickas, and Zary Segall. When Peer-to-Peer comes Face-to-Face: Collaborative Peer-to-Peer Computing in Mobile Ad hoc Networks. In *First International Conference on Peer-to-Peer Computing*, Linköping, Sweden, 27-29 August 2001.
- [7] Sun Microsystems. jxme: JXTA Java Micro Edition Project. <http://jxme.jxta.org/>, 2006.
- [8] J.S. Olson, S. Teasley, L. Covi, and G. Olson. *The (currently) unique advantages of collocated work*. MIT Press, 2002.
- [9] Rockyroad. jrta. <http://www.jrta.org>, 2006.
- [10] Tobias Straub and Andreas Heinemann. An anonymous bonus point system for mobile commerce based on word-of-mouth recommendation. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 766–773, New York, NY, USA, 2004. ACM Press.