Context-Aware P2P Middleware for Mobile Wellness Applications

JUSSI ALA-KURIKKA
Low amounts of physical exercise are known to expose people to many health hazards. Modern lifestyle involves a reduction in the amount of physical exercise we are required to take during our daily activities, which should be compensated with increased spontaneous exercise. However, people tend to make short-sighted decisions related to their way of life. In this work, we present a novel concept of a mobile application designed to motivate people to work out and to enable more efficient training. Sharing training programs and results within peer groups is at the heart of the concept. Potential users of the application include sports enthusiasts as well as professional athletes and their coaches. Development of a prototype of the application for the Series 60 smartphone platform is facilitated by utilizing a middleware of our own making. The middleware features peer-to-peer-oriented communication services with dynamic support for different communication protocols and connectivities. We discuss the architectures of both the application and the middleware at a detailed level, and compare the prototypes with existing research work and commercial applications. To gain knowledge of the feasibility of the concepts, we analyze the prototypes in various ways, including estimating the resource savings gained by utilizing the middleware and performing delay measurements. Furthermore, an end-user evaluation provides information about usability and utility mostly regarding the sports application. It was discovered that the user evaluation results were negatively affected by shortcomings of the prototype implementations. However, the concepts were found feasible in all of our tests and the novel features of the middleware prototype work as expected.

Keywords: wellness, mobility, middleware, peer-to-peer.

TIIVISTELMÄ


Avainsanat: hyvinvointiteknologia, mobiilinett, välikerrosohjelmistot, vertaisverkot.
# CONTENTS

**ABSTRACT**

**TIIVISTELMÄ**

**CONTENTS**

**PREFACE**

**ABBREVIATIONS AND SYMBOLS**

1. **INTRODUCTION** .................................................. 9

   2.1. Introduction to Distributed Systems .......................... 11
   
   2.1.1. General .................................................. 11
   2.1.2. Distributed System Models ............................... 12

2. **ENABLERS FOR MOBILE PEER-TO-PEER** ..................... 11

   2.1. Introduction to Distributed Systems .......................... 11
   
   2.1.1. General .................................................. 11
   2.1.2. Distributed System Models ............................... 12

   2.2. Peer-to-Peer ................................................ 13
   
   2.2.1. Centralized Architecture ................................ 14
   2.2.2. Decentralized Architecture .............................. 15
   2.2.3. Hybrid Architecture ..................................... 16
   2.2.4. Comparison of the Architectures ......................... 17

   2.3. Session Initiation Protocol ................................ 18
   
   2.3.1. Background ............................................. 18
   2.3.2. P2P SIP .................................................. 18

   2.4. Middleware .................................................. 19
   
   2.4.1. Software layers ......................................... 20
   2.4.2. P2P Middleware ............................................ 20

   2.5. Mobile Devices and Services ............................... 21
   
   2.5.1. Mobile Computing ....................................... 21
   2.5.2. Device Model ............................................. 23
   2.5.3. Context Awareness ....................................... 24
   2.5.4. Mobile P2P ............................................... 25
   
   2.5.4.1. JXTA for J2ME ....................................... 27
   2.5.4.2. Mobile Gnutella ....................................... 27
   2.5.4.3. Mobile eDonkey ....................................... 27
   2.5.4.4. Mobile Chedar ......................................... 28
   2.5.4.5. SIP Based Mobile P2P ................................ 28

3. **WELLNESS TECHNOLOGY** ......................................... 29

   3.1. HOTACT ..................................................... 29

   3.2. Health Club ................................................ 30

   3.3. Wireless Sensor Network in Sports ......................... 30

   3.4. Wellness Diary ............................................. 31

   3.5. Heart Rate Meters .......................................... 31

   3.6. FRWD ....................................................... 32

   3.7. Vivago Personal Wellness Manager ......................... 32

4. **PROTOTYPE DEVELOPMENT** ....................................... 33

   4.1. Concepts .................................................. 33
   
   4.1.1. PnPAP .................................................. 33
   4.1.2. Wellness Application .................................... 36

   4.2. Analysis of Wellness ....................................... 37
   
   4.2.1. Use Cases ............................................... 37
   4.2.2. Requirements ............................................. 38
   
   4.2.2.1. Documentation Style ................................. 38
This Master’s Thesis was done in the Application SuperNetworking (All-IP) project at the MediaTeam research group at the Department of Electrical and Information Engineering at the University of Oulu. I am grateful for the years I had the privilege of being able to work in MediaTeam. Thanks go to all the personnel at MediaTeam for creating an exceptionally warm working atmosphere with interesting research problems and professional colleagues.

All-IP was the last project I participated at MediaTeam. The project aimed at developing a foundation for All-IP applications, services and networks. I wish to thank the partners of the All-IP project: Tekes (Finnish Funding Agency for Technology and Innovation), Nokia, TeliaSonera, Elektrobit, Serv-It and IBM. Above all, I want to thank the personnel of the All-IP project, especially the Project Director Prof. Mika Ylianttila, the Project Manager Erkki Harjula, and researchers M.Sc. Timo Koskela, M.Sc. Otso Kassinen, M.Sc. Juuso Ohtonen and M.Sc. Douglas Howie, with whom it was a great joy to work.

I wish to deeply thank FRWD Technologies for our cooperation regarding the Wellness application, and SATS Heikinkatu at Oulu for sponsoring the end-user testing with free tickets to the gym and gymnastic exercises. Thanks belong to my Master’s Thesis supervisor, Prof. Tapio Seppänen, and PhD Pertti Väyrynen for providing expert language revision.

I also wish to express my gratitude to my friends and relatives, who have supported and encouraged me throughout this work. Last, but foremost, I want to thank my beloved wife Taru from the bottom of my heart, for her constant support and understanding throughout my studies, and especially during the intensive periods of working on this thesis.

Vantaa, 20 May 2007

Jussi Ala-Kurikka
<table>
<thead>
<tr>
<th>ABBREVIATIONS AND SYMBOLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2G</td>
</tr>
<tr>
<td>3G</td>
</tr>
<tr>
<td>3GPP</td>
</tr>
<tr>
<td>ACPC</td>
</tr>
<tr>
<td>API</td>
</tr>
<tr>
<td>BAN</td>
</tr>
<tr>
<td>BT</td>
</tr>
<tr>
<td>CORBA</td>
</tr>
<tr>
<td>CSF</td>
</tr>
<tr>
<td>DC</td>
</tr>
<tr>
<td>DCOM</td>
</tr>
<tr>
<td>DHT</td>
</tr>
<tr>
<td>DLL</td>
</tr>
<tr>
<td>DNS</td>
</tr>
<tr>
<td>DOLR</td>
</tr>
<tr>
<td>DoS</td>
</tr>
<tr>
<td>DSL</td>
</tr>
<tr>
<td>ECG</td>
</tr>
<tr>
<td>EDGE</td>
</tr>
<tr>
<td>ER</td>
</tr>
<tr>
<td>FP</td>
</tr>
<tr>
<td>FR</td>
</tr>
<tr>
<td>GSM</td>
</tr>
<tr>
<td>GPRS</td>
</tr>
<tr>
<td>GPS</td>
</tr>
<tr>
<td>GUI</td>
</tr>
<tr>
<td>HCon</td>
</tr>
<tr>
<td>HR</td>
</tr>
<tr>
<td>HTTP</td>
</tr>
<tr>
<td>IETF</td>
</tr>
<tr>
<td>IM</td>
</tr>
<tr>
<td>IMS</td>
</tr>
<tr>
<td>IP</td>
</tr>
<tr>
<td>J2ME</td>
</tr>
<tr>
<td>MIDP</td>
</tr>
<tr>
<td>MIT</td>
</tr>
<tr>
<td>MMS</td>
</tr>
<tr>
<td>MMUSIC</td>
</tr>
<tr>
<td>MP3</td>
</tr>
<tr>
<td>MPAA</td>
</tr>
<tr>
<td>MVC</td>
</tr>
<tr>
<td>NAT</td>
</tr>
<tr>
<td>O/S</td>
</tr>
<tr>
<td>P2P</td>
</tr>
<tr>
<td>PC</td>
</tr>
<tr>
<td>PDA</td>
</tr>
<tr>
<td>PG</td>
</tr>
<tr>
<td>PnPAP</td>
</tr>
<tr>
<td>Acronym</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>QoS</td>
</tr>
<tr>
<td>RAM</td>
</tr>
<tr>
<td>RFC</td>
</tr>
<tr>
<td>RFID</td>
</tr>
<tr>
<td>RIAA</td>
</tr>
<tr>
<td>RMI</td>
</tr>
<tr>
<td>RTP</td>
</tr>
<tr>
<td>S60</td>
</tr>
<tr>
<td>S.D.</td>
</tr>
<tr>
<td>SDK</td>
</tr>
<tr>
<td>SIM</td>
</tr>
<tr>
<td>SIMPLE</td>
</tr>
<tr>
<td>SLOC</td>
</tr>
<tr>
<td>SM</td>
</tr>
<tr>
<td>SMTP</td>
</tr>
<tr>
<td>SIP</td>
</tr>
<tr>
<td>SIPPING</td>
</tr>
<tr>
<td>SMS</td>
</tr>
<tr>
<td>SP</td>
</tr>
<tr>
<td>TCP</td>
</tr>
<tr>
<td>TTL</td>
</tr>
<tr>
<td>UDP</td>
</tr>
<tr>
<td>UID</td>
</tr>
<tr>
<td>UML</td>
</tr>
<tr>
<td>UMTS</td>
</tr>
<tr>
<td>UR</td>
</tr>
<tr>
<td>USB</td>
</tr>
<tr>
<td>UWB</td>
</tr>
<tr>
<td>WA</td>
</tr>
<tr>
<td>WCDMA</td>
</tr>
<tr>
<td>WG</td>
</tr>
<tr>
<td>WLAN</td>
</tr>
<tr>
<td>VoIP</td>
</tr>
<tr>
<td>WSN</td>
</tr>
<tr>
<td>WWW</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

Over the past decades, modern technology has greatly reshaped the way we live our lives. A significant portion of businesses today, or people in civilized countries, for that matter, would probably find it hard to keep going without the Internet. Computers are everywhere, and embedded ones are installed in our cars, household appliances, even clothes. In many ways, mobile Internetworking has already broken through as well, making services available anywhere at any time. Besides the profound effects on business and leisure, this has led to a decrease in the average amount of physical exercise we are required to take during our daily activities [1].

At the same time, the average age of people is growing. In many Western countries, this is due to large generations becoming older and increasing lifetimes, which adds to the number of people requiring medical care. In addition, modern medicine has not only become more effective, it has also become more expensive. To overcome this considerable financial problem, we would need to lessen the need for medical care somehow and start moving the focus from treatment to prevention. It may sound as if people would want this spontaneously. In practice, however, we tend to make short-sighted decisions related to our way of life [1].

It is a widely accepted fact that physical exercise can reduce many health risks, including overweight, diabetes and cardiovascular diseases, and help maintain strength, bone mineral density and many musculoskeletal problems just to mention a few. In the elderly, training improves flexibility and quality of life, and extends the ability of getting along without constant care [1], [2]. Thus, we need to find ways to strengthen the motivation of people to take exercise, and to do it in a right way. This work is about using modern technology to do exactly that.

We present the novel Wellness application, which is an electrical counterpart of a sportsman’s notebook. It can be used to plan physical exercises of many sorts and to record results during workout for later study. The application integrates with sensors to monitor and store several features of the user’s exercise, including heart rate and location. It guides the user during her exercise, telling her what type of exercise was planned to be done next, and helps her in staying within the exercise specific target heart rates much like a state-of-the-art heart rate meter would. On top of these functionalities, Wellness offers the ability to exchange this information in near real time in a virtual community of his preference, referred to as a peer group. A peer group can be formed between a user and his personal trainer, for example, or by a group of friends who have promised themselves to start working out regularly. The application aims at providing facilities for sharing useful information related to sports thereby making working out more effective, safe and hopefully more fun. Perhaps, it can also encourage a user’s wholesome competitive drive to further boost his motivation.

In this Master’s Thesis, we will develop a prototype of the Wellness application for Series 60 mobile phones. We will implement Wellness on middleware called the Plug-and-Play Application Platform (PnPAP) that has been developed as a joined effort by the researchers of the Application SuperNetworking (All-IP) project at the University of Oulu. The existing PnPAP prototype features already implemented high-level networking services for the S60 platform, which enables Wellness to be implemented with significantly less effort. However, as one of the key persons in designing and realizing PnPAP from the scratch, we will also
present how PnPAP was developed with higher specificity than what has been seen in earlier publications. In fact, we will present the development of PnPAP from the viewpoint of the Wellness application, and document PnPAP use cases and requirements based on the needs of such software. Later, we will go further to show that the same use cases and requirements, and thus any implementation fulfilling them, also serve other types of networked applications. Our presentation eventually aims at providing a model for both the middleware and the application which enables others to develop them further - perhaps even into a commercial product.

Furthermore, we will illustrate the performance of our prototypes and put forward statistics related to them. Basic software testing and measurements are complemented by an end-user evaluation performed in collaboration with two other Master’s Thesis writers having their own set of questions in the shared questionnaire. The evaluation provides qualitative information regarding the usability, feasibility and even the economic potential of our prototypes in addition to the quantitative information gathered otherwise. The results are then discussed to find the good and the bad in our specifications and implementations.

The work is broken down as follows. First, we will show the state-of-the-art related to the technologies that we require in developing our prototypes. The state-of-the-art, covering both published research and existing commercial applications, will serve both as a starting point for our work and as rationale for the choices we make later on. The survey is divided into two sections. The first one in chapter 2 deals with middleware, mobile networking and P2P. The second part in chapter 3 deals with wellness technology related to our Wellness application concept. Chapter 4 starts the discussion on our novel concepts and development of the prototypes. In chapter 5 we put forth the results of our work, including the design and outcome of the end-user evaluation. Chapter 6 concentrates on discussing the results and comparing them against the state-of-the-art. Questions not solved by our work and improvement ideas are also dealt with. Chapter 7 summarizes the work.
2. ENABLERS FOR MOBILE PEER-TO-PEER

2.1. Introduction to Distributed Systems

2.1.1. General

A distributed system can be defined as having networked computers running components that communicate and coordinate their actions by passing messages. The Internet and the Web are well-known examples of such systems, so are mobile networks, intranets, even home networks. In many distributed systems, the sharing of different kinds of resources is the main use case. Figure 1 shows the diversity of the term. First, the whole network of networks (Internet) is a distributed system. It is no doubt the largest distributed system that man has ever built. The independent networks within the Internet are distributed systems themselves. On the other hand, a shared resource can be calculation power on a server, a tune on an MP3 player, a printer at home, a file on a laptop at work, or an audio connection between two mobile phones. Thus, distributed systems are everywhere, and people are so accustomed to using them that they often do not even realize it. [3, pp. 1-3].

![Figure 1. Distributed networks.](image-url)

The Internet enables clients and services to run in a heterogeneous environment of different networks, computers, operating systems, programming languages and implementations. The differences are masked by using a set of common protocols, the Internet protocols. However, IP, even TCP and UDP protocols, do not address the representation of data in different hardware and programming languages. To achieve interoperability, the developers of programs need to agree to more specific standards on top of the Internet protocols. For example, the Web relies on HyperText Transfer Protocol, HTTP, for retrieving and sending documents between web browsers and servers. [3] pp. 16-18]. For many purposes, there is an abundance of mutually incompatible protocols available. Using different protocols thus often leads to lack of interoperability between systems.
2.1.2. Distributed System Models

Dividing responsibility between system components (applications, servers and other processes) and placing the components on computers in the network is the starting point of distributed system design and greatly affects the performance, reliability and security of the resulting system. Two main architectural model types are presented next.

Distributed systems can be categorized using architectural models, for example by analyzing the placement of its parts and their relationships. Two distinct models are the client-server model and the peer-to-peer (P2P) model. A simple way to start modelling systems is to identify server, client, and peer processes. The client-server model defines a rather simple division of responsibilities: the server provides services that the client invokes. P2P, on the other hand, includes processes that cooperate and communicate symmetrically to perform a task. [3, pp. 29-34], [4].

Client-server is the most common architecture of distributed systems. Client processes invoke operations on server processes, and the servers reply to the clients using messages. Servers may themselves be clients of other servers. For example, web servers are clients of the Domain Name Service (DNS), and search engines are servers for web browser clients but clients to other web pages. The client-server model scales poorly, however, but it provides a direct and simple approach. Centralization puts the responsibilities on one computer, which can only serve up to a certain number of simultaneous clients. For improving scalability, the shared resources would need to be distributed to a wider set of computers and network links. Even varying the model by increasing the number of servers does not completely solve the problem. [4, pp. 117-133], [3, p. 35].

Figure 2 illustrates the paradigm. Clients communicate only with the centralized server which holds all the resources. For example, if a client wants to share a file, the content has to be uploaded to the server before it can be downloaded by another. [5, p. 9].

![Figure 2. The client-server architecture.](image-url)
2.2. Peer-to-Peer

In the peer-to-peer (P2P) model, all of the processes have similar roles. The idea of P2P is to use the vast amount of resources on the participating computers to provide the service. Processing, communication load and data objects are distributed, and an individual computer holds only a small part of the whole database. Objects must be replicated in many peers for being able to provide service, even when peers disconnect from the system, which is inevitable in large systems. This makes the architecture more complex compared to the client-server model. [3, pp. 35-36]

P2P is enabled by the rapid development of desktop computers and the availability of broadband network connections. With millions of daily users joining for example the Kazaa, BitTorrent and Skype systems, P2P traffic sums up to 80% of all Internet traffic [6], [7]. P2P has also changed the distribution of this traffic: now the traffic is shared more evenly across the nodes, and more traffic is going on ‘at the edges of the network’. This sets new kinds of requirements on the network infrastructure. Asymmetric network connections such as the different Digital Subscriber Line (xDSL) technologies are not optimal for P2P because of their limited upstream bandwidth. [5, pp. 20-21].

Some argue that a clear definition for P2P is lacking. If, for example, I run a web server on my PC and browse pages provided by other servers simultaneously, does that account for P2P? Schollmeier provides one definition [8]:

A distributed network architecture may be called P2P network if the participants share a part of their own hardware resources. These shared resources are necessary to provide the service and content offered by the network. They are accessible by other peers directly, without passing intermediary entities. The participants of such a network are thus resource providers as well as resource requesters.

By this definition, it is questionable if some so-called centralized P2P systems are in fact P2P if they require a central point for bootstrapping. Some advanced P2P systems include firewall or Network Address Translation (NAT) traversal through intermediate peers when required. However, the key point to P2P is to share resources between peers, distributing the load of providing the service. To reduce the room for arguments, we claim that P2P requires integration of the two roles into one process. Therefore, running a web server and a browser on the same computer simultaneously is not P2P, unless the two functions are combined into a process. They could be, but it would make little sense. Conversely, sharing files could be done by uploading all files to a server where they would be available for download by others. A more natural way would be for the users to be able to get them directly from the source without intermediaries.

One of the most famous P2P systems is Napster, which provided file-sharing service for tens or hundreds of thousands of users until brought down due to intellectual property violations [3, pp. 35-36]. Napster and other P2P file sharing can be regarded as distributed file systems [3, pp. 323-366], although many are not transparent in the way they represent resources - e.g. files have to be expressly downloaded and shared. Despite its problems, Napster showed the feasibility of sharing files with P2P and ever since P2P has dominated the traffic in the Internet. [6]. In fact, communication in the Internet during its early days
was P2P as every node was running a server that others could connect. Many of
the original Internet protocols that are still in use are peer-to-peer such as the
Domain Name System (DNS) and Usenet news. [5, p. 6].

Thus far, P2P is known by the wider public for its illegal use in sharing copy-
right protected material. The media industry, especially the Recording Industry
Association of America (RIAA) and Movie Picture Association of America
(MPAA), has been fighting strongly against P2P and the distribution of pirated
material [9]. This has even lead to legislative changes, making it possible to sue
P2P developers for the copyright violations done with their software [10]. Some
studies point out that the media industry could actually benefit from P2P by
adjusting their business models and using P2P as a content distribution channel.

Forgetting the type of material exchanged in P2P networks for a moment, there
are many impressions of categorizing P2P architectures. For example, Lehtinen
[5, p. 9] divides P2P systems into structured and unstructured architectures,
and the unstructured ones further into three generations or categories, namely
centralized, decentralized and hybrid architectures. Lehtinen considers structured
architectures the ones that use the Distributed Hash Table (DHT) algorithm to
find resources in the network. All resources are named by giving them a unique
identifier (ID) or a hash. Each peer is responsible for storing a subset of the
whole hash space, with sufficient replication to account for disconnections, etc.
Pastry is an example. [5 p. 14]. We will discuss unstructured architectures more
closely.

2.2.1. Centralized Architecture

Napster was clearly a first generation P2P system that can be identified for having
a centralized resource indexing server, which resembles the client-server model.
Knowledge of shared resources is kept on the server. However, the resources
themselves are stored on the peers and transferred directly between them without
going through the server, which is the main difference to client-server. Lehtinen
[5, pp. 10-11] calls the indexing server a super-peer. We, however, reserve that
term for use in the third generation hybrid approach. For example, in Direct
Connect (and its open-source version, DC++) the indexing server is called a
hub, which we will use from now on to refer to an indexing server.

Figure 3 illustrates the concept. Peers register with the hub that listens for
queries sent by the peers (1). With the registration, the peers provide metadata
of the resources they share. The server searches its current database and replies
with a direct addressing means to the query result (2). Then, the originating
peer is able to send a download request directly to the peer that has a matching
resource (3) and download is started (4).

The first generation approach provides some advantages. It is lightweight to the
peers, thus it suites mobile environments well where resources such as networking
bandwidth are limited. It also provides authority through the indexing server. For
example, a network operator could provide a file sharing service for its customers
for an additional fee and control the material that is available [5 p. 11]. On the
other hand, the server requires administration and maintenance and is a single
point of failure.
2.2.2. Decentralized Architecture

The second generation decentralized architecture is also known as pure P2P. It offers greater scalability, anonymity and fault tolerance by removing the need for a centralized server altogether - all peers are truly equal. Bootstrapping to the P2P network is considerably more difficult in this architecture, as there are no static peers. A peer can have its own database of the known peers and/or IP multicast or broadcast can be used to find neighboring peers. In case at least one peer is found, the list of its peers can be downloaded and a connection to the P2P network has been achieved. [5, pp. 12-13].

An example search in a decentralized architecture is shown in Figure 4. The queries are forwarded from peer to peer with a limited Time-To-Live (TTL) value (1). The algorithm depends on the application, e.g. Gnutella 0.4 uses a flooded request algorithm that broadcasts the search message, whereas Freenet, on the other hand, uses a more intelligent algorithm. If a search result is found on some peer, a reply is sent, usually using the same path the request was routed along (2). Further communication (3 and 4) is done either directly between the peers or via intermediate peers, depending on whether strong anonymity and/or firewall or NAT traversal is needed. [5, pp. 12-13], [4, pp. 100-105].
2.2.3. Hybrid Architecture

Differing views exist on the definition of the third generation of P2P systems. We refer to the third generation as implementing super-peers in addition to regular peers [12]. The super-peers take over responsibilities, including message routing from regular peers. Typically, any peer can become a super-peer if it has enough resources and a good enough network connection. This definition would enable even very resource-constrained mobile devices to become part of the network as regular peers, whereas the strictly equal responsibilities between peers of a second generation system might prove too demanding for these types of devices.

Indeed, the third generation is a hybrid of the centralized and decentralized architectures. Hybrid networks have several Super-Peers (SP) that are equivalent to hubs, but they are interconnected and thus they themselves form a decentralized P2P network. An example of the hybrid topology is given in Figure 5 [5, p. 13]. The SPs typically maintain a database of the other SPs, peers that are connected to it and the peers' shared resources. However, they do not know about the peers connected to other SPs. Hybrid architectures are in popular use in peer-to-peer applications like Gnutella 0.6 [13] and Kazaa. Also, the massively popular Skype Internet telephony application from the developers of Kazaa uses the same architecture. A research paper with a more detailed description of the Skype architecture is given by Baset and Schulzrinne [14].

When starting a query, a peer sends it to its SP. The SP then forwards it to other SPs and if there is a match, the peers can communicate directly or through the SPs, depending on the architecture. Actually, for a regular peer, the hybrid topology is equivalent to the centralized architecture. In some 3rd generation architectures, however, peers can connect to multiple Super-Peers at a time. If resources, e.g. network bandwidth and processing capacity, allow, peers can also switch between the "regular peer" and "Super-Peer" roles dynamically. This clearly separates the third generation from the first one, where the two roles are implemented in different processes. [5, pp. 12-13].

![Figure 5. Hybrid P2P architecture.](image-url)
Another view of the third generation P2P by Coulouris et al. [3, pp. 399-401] who places Gnutella, Kazaa [15] and BitTorrent [16] on the second generation. Coulouris’ view is characterized by middleware layers for application-independent management of distributed resources. The third generation also adds guarantees of delivery for requests in a bounded number of hops and decides places for replicas of resources intelligently. According to Coulouris, the best-known examples include Pastry [17], Tapestry [18], CAN [19], Chord [20] and Kademlia [21].

In either case, many advanced P2P systems form their own message routing network on top of the IP network called an overlay. Overlay provides services not achievable by IP routing, including its own addressing which allows a very large and flat name space. Overlay enables security, even in environments with limited trust with a limited degree of anonymity. Also, updates of the routing table and replication of routes and objects can be done freely. [3, p. 401]. In addition to overlays, many P2P systems feature virtual communities that can be formed between peers. These communities are sometimes called Peer Groups (PG). They can have limited or unlimited access. The PGs form boundaries in which peers can work in. For instance, they can be used in computer-supported cooperative working where users form a group in which they can share resources such as files [3, p. 8].

2.2.4. Comparison of the Architectures

The different architectures provide differing properties. For example, the pure P2P topology has been found to have scaling issues with large numbers of peers. Still, it is the most failure tolerant, while the first generation systems cannot operate without a working hub that is also likely to become a performance bottleneck as well. The hybrid approach, on the other hand, scales quite well and has good resiliency. [5, pp. 14-15]. However, typically newer architectures are implementationally more complex than old ones. For the use these systems to be feasible in a software project, it may require an existing open-source implementation. When one is not available, an older and more simple P2P system can provide sufficient features with acceptable implementation cost.

The wealth of different P2P protocols is at least partly due to the little interest towards P2P by any of the official standardization bodies. The protocols have thus been proprietary and in many cases used by only one application. [5, p. 18]. As with client-server such as Instant Messaging (IM) protocols, the abundance of protocols has created a severe interoperability issue, since close to none of the protocols are compatible with each other. If two peers have different P2P applications running, they typically cannot interoperate.

The lack of standardization effort can partly explained by the relative freshness of popular P2P demand. In the beginning of its comeback, P2P was mainly used for illegal purposes. Thus, the protocols were designed to be able to dodge the administrators’ most typical ways of trying to restrict the P2P traffic, and the wide variety of P2P protocols was actually a positive thing in that respect. For legitimate purposes, on the other hand, this makes the situation worse. Some of the protocols have been accepted widely, however, which has made them de-facto standards, for example the FastTrack protocol originally used by Kazaa. It is now used by applications including Morpheus, Grokster and Apollon [5, p. 18].
2.3. Session Initiation Protocol

2.3.1. Background

The Session Initiation Protocol (SIP) is a general Internet signaling protocol for creating, modifying and terminating multimedia sessions between two or more participants. SIP can be used for Internet Protocol (IP) telephony, Instant Messaging, multimedia conferencing and similar applications. [22]. SIP was drafted by the Internet Engineering Task Force (IETF) Multiparty Multimedia Session Control (MMUSIC) working group (WG) in 1997. The SIP WG was established in 1999. Later, Session Initiation Proposal Investigation (SIPPING) and SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) WGs were set up for investigating further applications of SIP and defining IM extensions for it, respectively. [23]. One thing that makes SIP significant is that SIP was made the signaling protocol of choice for the IP Multimedia Subsystem (IMS) in 3rd generation mobile networks. IMS is a part of 3G from 3GPP Release 5 onwards. [5, pp. 22-23].

SIP is a text-based protocol based on the HyperText Transfer Protocol (HTTP) and Simple Mail Transfer Protocol (SMTP). SIP provides different aspects of session establishment, namely user location (address), availability, capabilities, establishing session parameters and session management, including transferring, modifying and terminating the session plus invoking services [22]. SIP is lightweight in comparison with the other major Internet signaling protocol, H.323. SIP can be run on many transport level protocols, for example TCP and UDP. SIP provides only signaling services, thus actual communication includes additional application-level protocols such as the Real-Time Protocol (RTP) for audio/video transmission. [5, pp. 22-24].

SIP is an end-to-end protocol. SIP messages are routed from the originator through SIP proxies to the target node. SIP can be thought of as hybrid peer-to-peer, where SIP proxies are Super-Peers and they form a network of their own. However, like in the 1st generation P2P, a regular peer usually cannot become a proxy dynamically. During a single SIP transaction, client-server topology is used: a request sender acts as a client and the target node that sends a reply acts as a server. [5, pp. 22-24].

2.3.2. P2P SIP

In practice, SIP requires an infrastructure (of proxies etc.) that users need to connect to. End-users may consider setting up a SIP client to connect to the infrastructure too difficult, as the clients rarely configure themselves automatically. Also, firewalls and NATs often pose problems requiring additional solutions to go around that add to the setup and administration burden.

An example of the opposite is Skype, which requires nothing but installing the Skype client to work. The popularity of Skype and similar non-SIP Internet telephony services has awakened a concern within the IETF, which is now seeking to make SIP more attractive. By defining additional specifications for P2P-type SIP that are compatible with the original SIP, the proposed new P2PSIP Working Group (WG) of the IETF [24] aims at providing standardized services with
features and user friendliness matching the existing P2P Voice over IP (VoIP) services. Other popular P2P services such as file sharing are not within the scope. P2P-SIP also targets to enable using SIP and SIMPLE protocols in situations where there is no infrastructure available, such as when not connected to the Internet. Regular SIP was not designed to operate in environments such as this.

Besides the P2PSIP working group, there are other interesting examples of SIP being used for P2P. SIPshare by EarthLink is an example of a file sharing application that uses SIP as the signaling protocol with the pure P2P topology. SIPshare is implemented for the desktop environments, however. No implementation suitable for mobile terminals is available. Lehtinen presents a simple SIP-based P2P application in [5]. The application has been prototyped on the Series 60 platform. Lehtinen also provides measurement results, which show that the application is feasible even on current technology.

2.4. Middleware

Although distributed systems may seem simple, there are many challenges in building them. One option is to solve all the problems within the application. However, sometimes that requires too much work, and also the reusability of such programming may not be sufficient. Splitting the development in parts small enough may help in assessing the required amount of work. It would also be convenient if some components of the software already existed, and we could reuse modules implemented in a previous project or by someone else.

This is where different kind of application frameworks come in. Although operating systems evolve and today’s programming environments provide both O/S services and some services implemented by themselves, these two usually stay as general as possible and do not keep up with the latest trends. For instance, in case of communication protocol libraries supplied with a programming environment, in many cases only one - perhaps proprietary - is shipped. Different kind of application frameworks exist that solve one or more challenges to make them easier to handle by the application programmer.

For example, many frameworks include means for providing some type of security, which is often regarded as an important feature. Security can actually be divided into three distinct components. The first is confidentiality - not disclosing information to unauthorized people. The second is integrity, which is about making sure that information is not altered or corrupted. The last component deals with the availability of resources: protection against interference with the means to access the resources. Denial of Service attacks (DoS) strive at breaking that protection. All the aspects of security are especially important in electronic commerce, banking and healthcare. [3, pp. 18-19]. Frameworks typically offer services that are tested for scalability, which means acceptable operation when the amount of services and users significantly increases. For example, responsiveness to user commands should be adequate in all cases. Failure handling, on the other hand, is about dealing with the failure of some components of the distributed system. [3, p. 43]. The partial nature of the failures makes them difficult to handle. Also, the distributedness of the systems allows simultaneous actions. Concurrency has to be dealt with; otherwise, operations may conflict with each other and produce inconsistent results. [3, pp. 19-23].
2.4.1. Software layers

Software architecture refers to the structuring of software as layers or modules, or, more recently, in terms of services offered by processes running locally or remotely. This view of software is shown in Figure 6. The lowest level hardware and software are commonly referred to as the platform. Like the upper layers, these low-level layers provide services to the layers above them. The services can be implemented independently in each computer because of the common interfaces between layers. The platform usually serves up to the level of basic communication and process coordination facilities. Intel x86 with the Microsoft Windows operating system is one example platform.

![Software layers diagram]

Figure 6. Software and hardware service layers.

As shown in Figure 6, middleware means a software layer that provides a programming abstraction including masking the heterogeneity of the underlying environment, such as networks and operating systems. Middleware is represented by processes or objects in computers that implement communication and resource-sharing support for distributed applications. Middleware provides building blocks and raises the abstraction level of the communication activities for software components that cooperate in a distributed system. Remote method invocation, event notification and communication between processes are commonly provided services. CORBA, Microsoft’s Distributed Component Object Model (DCOM) and Java Remote Method Invocation (RMI) are examples of middleware, most of which is implemented over the Internet protocols, which provides abstraction for underlying networks. However, middleware extends the abstraction by hiding differences in operating systems and hardware. Middleware can also hide the distribution of components: the programmer does not then have to differentiate invoking methods of objects running locally or remotely.

Again, middleware provides services to the layer on top of it, i.e. application programs. For example, middleware can provide services that help in overcoming one or more common challenges in building distributed system which simplifies the implementation. However, it is very difficult to solve some problems without support at the application level. Thus, all communication activities probably
cannot be abstracted away from the application with middleware. The argument is that correct behavior depends on checks that only the application can do because of the data it holds alone. [3] p. 34].

2.4.2. P2P Middleware

One of the key elements to P2P is to provide a means for clients to access resources quickly and dependably wherever they are in the network. The techniques are usually specific to each system, first generation ones using centralized indexes and second generation relying on fully distributed tables. P2P middleware systems aim at solving this issue by providing services for the placement and finding of distributed objects. [3, pp. 404-405].

A profound requirement for P2P middleware is to make the construction of distributed services easier. Therefore, it must enable peers to locate and communicate with each other and to share any type of resource within the system. The middleware should have a simple programming interface that also serves adding and removing resources and hosts at will. [27] also lists some other requirements. First, middleware should be designed to accommodate global scalability meaning hundreds of thousands of users. That requires effective load balancing which is enabled by random placing of resources with enough replication. On the other hand, resources should be nearby, thus the requirement of minimizing the 'network distance' to heavily used resources. An important requirement is the one of supporting highly dynamic host availability. In large systems (whether or not P2P) hosts may usually come and go, but a P2P system should provide access to resources nevertheless. Lastly, anonymity and security, e.g. through authentication and encryption, must be supported. [3, pp. 405-406].

The requirements make it infeasible for all clients to maintain a database of all the available resources, thus, the knowledge is distributed. Each peer maintains a database of only a portion of the namespace. Each item is also replicated in as much as 15 other peers. All of this is handled by an algorithm called routing overlay within the middleware. For example, Pastry uses an algorithm called Distributed Hash Table (DHT), whereas Tapestry uses Distributed Object Location and Routing (DOLR) method. [3, pp. 406-409].

JXTA [28] is one more example of a P2P middleware. Implementations of the open source middleware are available for different platforms, but the most popular version that is developed the most actively is based on Java. JXTA provides many layers of services, ranging from pipes, which are basically equivalent to sockets, to more advanced services. JXTA is natively a 3rd generation P2P system.

2.5. Mobile Devices and Services

2.5.1. Mobile Computing

When developing mobile distributed applications, mobility is a key factor which cannot be ignored. The services have to be designed from the start or adapted to optimally suit the environment they are to be operated in. Mobility requires a whole new paradigm for designing applications: it offers totally new use cases,
but also demands a lot from the developer. Mobility usually translates to limited resources including small display and input devices, limited battery, processing power and memory, even if the technology has evolved rapidly and is continuing to do so. Especially, today’s mobile phones have got high-capacity removable memory cards and a respectable set of connectivities with color screens. IP traffic, Voice over IP (VoIP), for example, is also becoming a standard in mobile networks through the adoption of 3G standards. This subchapter deals with some of the most important challenges in developing mobile services.

As devices have become smaller, we have started to carry some with us or even wear them. Current mobile phones can be used as MP3 players and they provide constant access to our emails and the Web. Mobile services should exploit the (typically wireless) connectedness of those devices which enables us to connect the devices not only to one another but also to conventional desktop and server machines. The development of technology has also enabled the integration of intelligent devices into everyday environments and devices such as home appliances. This is called ubiquitous (i.e. pervasive) computing where computers psychologically ‘disappear’ by becoming an inseparable part of appliances and our daily lives. Typically, ubiquitous and mobile computing is regarded separate. However, at best, the collaboration of all of these devices would enable really interesting user scenarios. [3, p. 658].

Mobile computing started off in the 1980s with personal computers just light enough to carry, with telephone modem connections to other computers. Today, laptops with far greater resources have wireless connectivity through WLAN, Bluetooth, GPRS or 3G. A parallel development path has produced even smaller mobile devices known as handhelds, for example mobile phones and Personal Digital Assistants (PDAs). A trend has also been to gradually merge different handheld devices including mobile phones, PDAs and other handhelds such as digital cameras. Handhelds have blurred in distinction to larger computers by getting ports of their operating systems such as Linux and Microsoft Windows (Smartphone being the mobile version). Symbian is an exception - it has been developed for mobile terminals from the start. The set of short- and long-range connectivities can be the same independent of the device type. [3, pp. 658-659].

[29] lists two major problems related to building mobile wireless systems. Due to bandwidth and power constraints, direct communication between two given mobile nodes is most often not available. The first problem is that mobile devices should maintain continuous connectivity when moving through regions of wireless coverage provided by base stations. This kind of a vertical handover between two base stations is depicted in Figure [7]. This dynamic environment cannot provide for ever-lasting error-free connections. Another connectivity with ‘downgraded’ bandwidth may become the only option to resume connection at any time, or the connection may be lost altogether for an undefined period of time. This has to be taken into account in service development. Second, devices should be able to communicate even in the inexistence of an infrastructure. The second type of solutions are called ad-hoc networking, which is exemplified by Figure [8]. [3, p. 659].

The basis for all security - trust - is often lowered in mobile systems because the involved parties may not have prior knowledge of each other or a trusted third party. Users, however, expect security (confidentiality, integrity and availability). Users are also concerned of their privacy, i.e. being able to control the accessi-
Figure 7. A vertical handover in an infrastructure of two base stations.

Figure 8. Ad-hoc networks.

ability of information about themselves. Mobile systems provide a suboptimal environment for conventional security protocols because of their restrictive characteristics. For example, public-key cryptography might be out of the question because of insufficient computing resources and disconnected operation is likely to occur. [3, p. 696-698].

2.5.2. Device Model

Mobile and ubiquitous computing has brought a new class of devices as a part of distributed systems. The devices are limited in energy, because portable devices usually run on batteries. Wireless communication is especially energy-intensive. Thus, energy-efficiency and device failure due to an empty battery need to be considered in design. Their computational resources are limited in terms of processor speed, work memory and persistent storage capacity and network bandwidth. Algorithms need to be designed to still finish in reasonable time. The devices can also be designed to utilize the resources in their environment. To make the devices context-aware, they need sensors and actuators. Sensors provide measurements of the physical world to software - position, orientation, temperature and sound, for instance. Conversely, actuators are software controllable devices that affect the physical world, for example curtain control or air-conditioning controller. Today, even tiny 'motors' that sense their environment and mobile phones can have a multitude of sensors such as temperature and position sensors like GPS, accelerometer and a camera. A camera, on the other hand, can be used for example to read a barcode of a product. [3, pp. 663-664]. A recent release from
Nokia has brought the N95 mobile phone to the market. The N95 is one of the most advanced mobile phones today, featuring an integrated GPS with a map application, a 5.1 megapixel camera, an inbuilt web browser and advanced media representation capabilities with state-of-the-art connectivity.

Mobile connectivity technologies differ in nominal bandwidth and latency, energy costs and whether they bring financial costs to the user. Costs may be substantial although some mobile operators have introduced flat-rate GPRS services. On top of that, mobile network operators may require having some level of control of the services that run in their network, and limit unwanted traffic. Many mobile devices feature many wireless connectivities such as WLAN, Bluetooth, GPRS, etc. In any case, frequent connections and disconnections and changing Quality of Service (QoS) has an impact on the system properties - the features of the communication may change considerably at run-time. All wireless connectivities tend to have longer delays, lower bandwidth and higher packet loss than fixed ones. Connections are also more prone to disconnections that last for an undetermined amount of time. [5, pp. 40-41]. What if the device has many connectivities, GPRS, WCDMA and Bluetooth, for example? Different access methods have properties including error rate, bandwidth and delay characteristics that can differ by orders of magnitude. Using them optimally brings us to the research problem of connectivity management which includes both horizontal (connectivity) and vertical (same connectivity with different base station) handoffs. The latter are discussed in great detail in [30].

Typically the input system is a small keyboard or a touch-screen display. Screen sizes of mobile phones are remarkably smaller than even the smallest modern PC displays. These features will remain limited because the devices must fit inside a pocket therefore limiting their physical size. Resolutions and the amount of colors have recently gotten better, but fitting all the useful information on the screen may still be difficult. [5, p. 41].

The Symbian operating system and Nokia’s Series 60 smartphone platform is one of the three major mobile phone platforms today. Symbian derives from Psion’s EPOC operating system which was originally designed for mobile devices. Thus, it provides resource efficiency. It is also popular, and the development community using the platform is vast. There is therefore an abundance of information available in the Internet. The relatively immature development environments can be regarded as a downside. Microsoft Smartphone and different mobile ports of Linux are the major competitors.

2.5.3. Context Awareness

To more tightly couple with the physical world, computers need ways to interact with it. They need architectures for processing information collected from sensors, and context-aware systems that respond to the sensed circumstances. Generally, the context of an entity (person, place, or thing, whether electronic or otherwise) is an aspect of its physical circumstances that are relevant to system behavior. Examples of context include location, time, temperature, the identities of the user and the ones nearby, and states of objects such as devices. Context can be reacted upon with rules: "if the user is in a meeting then reject calls automatically, unless the person calling is invited to the meeting", for example. [3] p. 683].
Context is determined by sensors which are combinations of hardware and/or software. Examples include satellite navigation in the form of GPS units to provide global coordinates and velocities or the use of WLAN or Bluetooth for location sensing, accelerometers to detect movement, magnetometers and gyroimeters to provide orientation. Additional examples are thermometers, light intensity sensors, microphones and sensors that measure physical load to detect whether a person is sitting on a chair or walking across a floor, Radio Frequency IDentification (RFID) readers and tags, and software to detect key presses on a computer. Sensors can be integrated as wearable computing which is about the devices that we carry attached to or within the fabric of our clothes or worn like heart rate meter bands, watches or even jewellery. Often, these specialized devices operate without user involvement. An early example is a personal active badge whose proximity makes other devices in an environment react somehow. For example, doors could be opened and lighting and air conditioning fine-tuned according to the badge owner’s preferences. [3, p. 660].

Reacting to the badge exemplifies context-aware computing which is an important subfield of mobile and ubiquitous computing. Context can make systems automatically adapt their behavior, context being anything that can be measured or detected directly or indirectly. [3, pp. 660-661]. For example, a mobile phone could ignore an incoming call if the user was in the middle of an intense workout at a gym. However, the user might want to receive calls before and after the workout, even if he still is at the gym. Hence, many types of context information would be needed. The Oxygen project at MIT [31] is one of the most well-known examples of complex context-aware systems.

Mäntyjärvi presents a procedure for recognizing context in mobile devices with the help of integrated sensors [32]. The work describes several context information sources in mobile environments and presents examples on their possible use. The procedure is demonstrated with experiments and mobile applications. Korpipää introduces a framework for acquiring, processing and distributing context information gathered from sensors [33]. The framework aims at enabling rapid development of context-aware applications for mobile devices.

2.5.4. Mobile P2P

The ways of conveniently sharing content in the mobile domain are currently limited. The content needs to be uploaded to a server either manually or with the help of a third-party application, or it can be sent via Bluetooth, Email or an Instant Messaging program, for example. Mobile P2P enables sharing content without uploading it to a centralized server. This is particularly useful if it is uncertain whether anyone will ever download the content. Unnecessary uploading is thereby eliminated. On the other hand, a file then has to be sent over the air interface every time another peer requests it - this may become a problem if the file receives popular attractivity. [5, p. 46]. Content replication and/or caching on a Super-Peer may provide a solution.

Mobile peer-to-peer is an enhancement to fixed P2P but hardly a replacement. It is likely that mobile P2P will be used when the user is on the move and when a fixed network is not available. There may be more self-created content like photos or videos shot with integrated cameras on mobile phones than in the popular
fixed P2P networks. The users may want to share the content with differing sized audiences, even the world. Past examples include disaster videos and funny videos. Piracy probably is not a large problem yet because of the limited number of users of mobile P2P. As the persistent memory capacity already is sufficient and mobile terminals include audio/video capabilities with MP3 decoding support, it is likely that at some point the sharing of music and video material illegally becomes an issue unless it is solved beforehand.

Lehtinen argues that searches in P2P are more convenient than in the World Wide Web since the searches in P2P networks span through the whole network. In the WWW, content is scattered across multiple servers making searching more difficult. If there were only a very limited number of P2P protocols, preferably only one, this would be the case. At least thus far there has been no general consensus on a standard P2P file sharing protocol. Instead, there is a wide variety of noninteroperable protocols. As searches thus span only one P2P network amongst many, P2P currently does not offer the ultimate solution. It must be noted that the number of existing P2P protocols is by multitudes smaller than the number of servers in the Internet, however.

Thus far, significantly less research has been done on P2P technologies suitable for mobile than fixed, but such studies are slowly emerging. This is probably related to the relatively recent introduction of truly open platforms on mobile devices. Mobile phones, for example, have become open only a few years ago in terms of allowing more than simple games to be installed after shipping. It appears they lack the same broad set of available development environments and well-documented APIs. However, the situation is getting better.

Pricing based on the amount of transfer does not encourage users to share files since both received and sent bytes affect the pricing. The mobile P2P service would need to provide you with some form of personal download credits for letting someone else download a file from your terminal to make sharing rational. Also, some form of intelligent distribution in the selection of a download source would be needed to lessen the risk of everyone downloading files from one specific peer, which would get costly to that peer. Of course, users may bear some costs if a service is otherwise valuable and attractive. In the case of a mobile P2P file sharing service, the value would potentially increase by the amount of users and shared content. All in all, the most straightforward way would be to minimize the connectivity costs.

Most P2P protocols and their reference implementations run on fixed environments, which makes implementing them more difficult on mobile devices. Many protocols have begun their life as proprietary, and they have an implementation only for PCs. That may rule those protocols out of the question for mobile development. If an open-source implementation or a library is not available, one needs to know the details of the protocol to be able to produce an interoperable implementation. Also, some protocols incur too much load on every peer, which is the case for pure P2P protocols, such as Gnutella 0.4.

The general consensus seems to be that pure P2P is infeasible for current mobile devices. Centralized or hybrid architecture or some kind of mix of those two is considered the optimal choice for mobile networks. To sum it up, most of the signaling traffic should be handled by Super-Peers (or similar) while mobile nodes would be responsible for its own file transfers and communication with its Super-Peer. Thilliez et al. suggest hybrid architectures for proximity applications where
devices communicate over short distances [34]. They argue that the architecture adapts well to changing environments and makes management easier thanks to partial centralization through Super-Peers. [5, pp. 50-51].

Practical mobile P2P research has been done for both realtime and non-realtime communication. Realtime P2P has been mostly applied for VoIP whereas non-realtime research has focused on file sharing. We will introduce some technologies in brief.

2.5.4.1. JXTA for J2ME

The JXTA community [28] has presented mobile versions of their core protocols under the name of JXME (JXTA for J2ME). An implementation of the protocols is available at their website. At the time of study, the protocols were restricted and had fallen behind from the development of the standard JXTA protocols. Only text messages could be sent with the demo applications, and file sharing was not supported, for example. The same restrictions applied for JXTA-C, which is a version of JXTA written in standard C language.

2.5.4.2. Mobile Gnutella

Hu et al. argues that conventional peer-to-peer file sharing networks are not suitable for mobile environments due to their bandwidth consuming nature. They propose a modified architecture for Gnutella with agents running on the fixed network that operate on behalf of the mobile devices. The agents act as normal Gnutella peers. The mobile terminal and its agent communicate e.g. the list of shared files using a lightweight protocol. The agent therefore handles most of the signaling traffic including searches. File downloads can be processed by the agent or the mobile node. Mobility support is provided through the fixed agent: the phone’s IP address may change rapidly but the network does not have to take it into consideration as long as the agent is kept aware of the mobile terminal’s current IP. [35]. The approach by Hu et al. resembles hybrid technologies. The constrained mobile devices can join in by using the same protocol but by handing over the excess signaling to a "Super-Peer". This approach is chosen by other mobile P2P architectures as well.

2.5.4.3. Mobile eDonkey

The original eDonkey protocol has been extended by Oberender et al. to support mobile P2P [36]. The proposed architecture contains a modified Index Server and introduces two new peers: Crawling Peer and Cache Peer. The Index Server tells the Cache Peer how popular each file is to optimize the cache contents and redirects requests to the Cache Peer when it contains the queried content. The Crawling Peers link the Index Server with other Index Servers in the Internet. The modified architecture results in a merge of centralized/hybrid architectures. [5, p. 49].

Firstly, the architecture provides better support for mobile P2P because, thanks to the Cache Peer, popular files can be stored in the fixed network. When a file
download is requested, the amount of needed file transfers over air interfaces is reduced by half to one. Secondly, mobile terminals need to connect to only one Index Server because the Crawling Peers enable all Index Servers to contain the same information. Normally, peers need to connect to multiple Servers to maximize their search effectiveness, which may be infeasible for a low-end mobile terminal. [5, pp. 49-50].

Oberender and de Meer present content replication as a means to improve performance and reduce traffic over wireless connectivities in another paper [37]. They find that traffic between mobile phones always passes at least one node in the core network, even if the mobiles are in the same cell. Thus, content should be cached near the core network nodes. [5, p. 50].

2.5.4.4. Mobile Chedar

Kotilainen et al. discuss Mobile Chedar, a mobile P2P middleware, in [38]. Their vision of mobile P2P focuses on short-range communication using Bluetooth. They present their J2ME based middleware prototype with a mobile P2P learning environment application.

2.5.4.5. SIP Based Mobile P2P

SIP and IMS based P2P research has gained attention recently. Akkawi et al. present a mobile gaming middleware based on SIP [39]. The middleware provides features including peer-to-peer connectivity, instant messaging and QoS by using services offered by IMS. The game clients communicate with a server by using standard SIP messages. Another architecture was presented for mobile P2P over IMS and SIP by Beijar et al. in [40]. The architecture is wisely divided into separate processes in order to support resuming file downloads even when the GUI is shut down, for instance.
3. WELLNESS TECHNOLOGY

Wellness can be considered the personal well-being of a person - his quality of life. Wellness technology, on the other hand, deals with goods, devices and technical apparatus whose purpose is to improve the well-being of a person. Its scope is broad, ranging from ambulatory through hospital devices to a variety of household gadgets and sports equipment. It includes artificial limbs, wheelchairs, devices that measure your blood sugar level and blood pressure meters. In this work, however, we limit the scope solely on sports technology, meaning the kind of apparatus that are used by athletes, keep-fit enthusiasts, coaches and personal trainers to enable planning trainings and guiding the trainers and tracking performance both during and after sports activities.

It is widely agreed that for health reasons it is advisable to stay fit by means of regular physical effort. Modern jobs typically are not physically demanding, and it is thus increasingly recommended to do regular workouts. [41], [1]. It has a positive effect on preventing many kinds of national diseases, especially cardiovascular diseases that are among the most important causes of death. Health of the people also has an impact on the health care costs. Jogging, cycling, not to mention group sports and the like, are therefore popular for a good reason. The problem is, however, that performing the same type of exercise can make it self repeating and boring. [41]. The solution may be to expand from performing just one type of sports at a time to having different regular workout activities. Personal goals and reaching them, effectiveness of the training or just the good feeling after the workout may also be sources of motivation. The effectiveness can be improved by help from books, personal trainers and different kinds of devices. We will discuss examples of related research and applications next.

3.1. HOTACT

Fitness machinery is constantly being made more attractive. One way is to provide more variety and a certain entertainment value by adding electronics for score-keeping - possibly for competing against oneself or someone else. The devices can offer different programs or display different information such as pulse rate, energy consumed or watts delivered. They can even provide simulated competitions on a screen where one of the participants moves at a speed relative to the training effort. [41].

An interesting prototype from as early as the year 1990 is given by Maurer and Soral in [41]. They present a model combining computer technology with fitness machines (bicycle home trainers, for instance) to obtain fitness training, learning and entertainment all at the same time. In their model, a motion oriented fitness machine called HOTACT such as a bicycle home trainer, a rowing machine or a running belt is used to control a computer that shows sequences of pictures with textual descriptions of the route taken. This kind of virtual travel enables the trainer to choose routes and from a wide range of picture material, providing needed variety. The virtual tours can be along cities, countries, regions - even the solar system or the blood vessels of a human being to learn about anatomy. They also implemented a prototype on a widely available bicycle home trainer. The model can be thought of as either staying fit with added motivation through
entertainment or, conversely, making learning more attractive by adding the fitness component. According to Maurer, HOTACTs could be deployed in fitness studios, resort places, homes, for heart rehabilitation, or in exhibitions.

3.2. Health Club

Pirttikangas et al. present an interesting concept of a context-aware health club in [42]. The application is based on a generic platform for context-aware applications. In the paper, they present a scenario where the system monitors the user in a cycling exercise. The context-aware terminal shows the exercise plan, and a heart rate meter is used during training to record information such as heart rate and speed of the bicycle. The information can only be downloaded to the system after the exercise, so feedback during the training cannot be given. The system includes tools for analyzing the data with a possibility of comparison between different exercises.

3.3. Wireless Sensor Network in Sports

Hyvönen presents a model for a wireless sensor network in intelligent sport environment in [43]. The model consists of two parts: first, an acquisition network that transmits information from sensors via the base station to a database on a server, and second, a distribution network, that provides users access to the information on the database. The acquisition network is further divided into a Body Area Network (BAN) and a support network (with base stations). In the work, a deployment plan based on the network model was presented for the Vuokatti sports center in Finland.

Hyvönen’s work is based on interviewing sport coaches to find what kind of information needs to be recorded. In skiing for example, heart rate and velocity need to be monitored in realtime, whereas altitude difference and distance can be monitored with a delay. The athlete’s performance defines the ranks in many sports, and often performance relates to time. Thus, measuring time is important, and with location, other variables such as speed can be derived. Also, measuring the physiological condition of an athlete through heart rate and/or lactate gives added value.

The model is designed using the client-server paradigm. Each system based on the model thus requires administration, and to satisfy that the model introduces various roles, including administrator, service desk person, maintenance person, and sports event organizer. The actual end-user roles of the system are athlete, coach and spectator of a sports event.

The BAN controller gathers the information from all BAN sensors and (possibly after some processing) forwards it to a base station using the support network. The idea behind BAN is to reduce the amount of devices connected to the long-range support network to prevent overload with many sensors. The BAN may also cache information between transmissions using a TinyDB database.

In the model, standard technologies were preferred. IEEE 1451 smart transducer standard was selected as the reference for sensors. Bluetooth, ZigBee and several UWB technologies were evaluated for the BAN, and of these UWB and
Bluetooth were suggested. Wi-Fi (IEEE 802.11 series), HiperLAN, WiMax, UWB and cellular network technologies, including GSM, GPRS, EDGE and UMTS were presented for the support network. In an scenario with large distances, Hyvönen suggests using GPRS.

Hyvönen also makes several suggestions for the distribution network. MySQL, PostgreSQL and Oracle are presented as the alternatives for the server database. She claims that the database should store information with as little redundancy as possible - meaning that derived information is calculated on every request. End-user devices include PDAs, mobile phones and laptops, as well as desktop computers. Presented security options (to provide access control) include passwords (one-time or multi-time), hardware token, symmetric key or public key based solutions and biometric authentication.

3.4. Wellness Diary

Wellness Diary is an S60 application for storing and analyzing wellness-related data collected from everyday life. Wellness Diary can be used for monitoring, tracking and analyzing weight, exercise, number of steps, eating management, fat-%, sleep and more. However, it does not include sensor support so information must be input by hand. The application does not provide guidance during a physical training session, but targets to track wellness on a more general level. It is available to download for free.

3.5. Heart Rate Meters

Probably the most well-known and important example of a sports tool available to ordinary people that improves the effectiveness of a workout is the heart rate meter. A modern model with a wireless ECG transmitter belt was presented by Polar Electro Ltd. in 1977. The meters quickly became popular among professional athletes, but in a few years cheaper models were presented that were available at hobbyist-level prices. Recording features were presented in 1984, which enabled computer based analysis after performance. Cheaper models have found their way to the market and features have developed greatly since.

Heart rate meters provide constant feedback during a physical training and thus enable accurately controlling the amount of effort. Other methods, including palpation method by counting the amount of beats manually and multiplying to get beats per minute, are prone to more or less error. This can lead to under- or overestimation of effort. Knowing the heart rate accurately makes for a safe, effective, and fun training. Meters can be used in all kind of sports independent of the level of professionalism, including gym training, walking, running, orienteering, cycling, ice hockey, and football to get detailed information and control of the training: intensity of the training with aerobic/unaerobic zones, the level of burning fat etc. Nowadays Polar is only one manufacturer among many to offer heart rate meters. New kinds of applications have also been introduced. For example, the Polar Team System enables coaches to monitor and record the heart rates of his
team. Suunto is one of the competitors for Polar.

3.6. FRWD

Technology evolution with minituarization has enabled interesting integration of devices. FRWD Technologies claim they sell the most advanced sports computer in the world. FRWD devices measure, record and show different properties of the user’s exercise, including location and speed using a built-in GPS sensor, heartrate from a heart rate transmitter belt, duration of workout, air pressure, altitude, and temperature. Currently, they have two new series of devices: the B series for real-time monitoring of exercises using a S60 mobile phone, and the W series that features a wrist display similar to a Polar or Suunto wrist meter. In addition to real-time monitoring, the details of each second of the exercise are stored in the internal memory that can be accessed using a PC Replayer software. This enables users to "relive their best sports moments" using computer simulation. For example, the route of an exercise can be shown using Google Earth.

3.7. Vivago Personal Wellness Manager

The Vivago Personal Wellness Manager is interesting in that it is not used only during exercise. Instead, it should be worn all the time. It measures the amount of physical activity and compares results with previous periods. It also monitors the amount of sleep and total calorie consumption. The meter performs measurements without a separate measurement belt. The idea of the device is to provide constant tracking of the most important aspects of wellness to see if and how the parameters have changed recently.
4. PROTOTYPE DEVELOPMENT

This chapter starts detailed discussion on our contribution, the Plug-and-Play Application Platform (PnPAP) middleware that has been developed as a joint effort in the All-IP project at the University of Oulu. The novel Wellness application will serve as a special application case developed in this Master’s Thesis. PnPAP has already been presented in several international publications. However, this is the first discussion at this detail level. We will virtually start our presentation from scratch, going through detailed documentation of the designs and prototypes with justifications for the choices we have made. The documentation can serve as a model for future middleware and applications, and anybody with adequate programming skills should be able to implement their own customized versions of PnPAP and Wellness. The most detailed focus will be on the PnPAP subcomponents developed by the author and the services required by the Wellness application.

As with any application framework, the services provided by PnPAP should be based on real application needs to maximize their value. To maintain a reasonable scope for this work, we will not collect the requirements from a large set of networked real-life applications. Instead, we will start off documenting our software specification with the Wellness application and then derive the requirements for PnPAP’s services from the perspective of that application. Later, however, we will show that the resource-sharing services of PnPAP that satisfy the set requirements are in fact very generic, and PnPAP can serve a variety of applications.

We will first present the high-level ideas behind both. Through detailed analysis of the concepts, we gradually increase our understanding of what we will implement and what is required from the implementations. During design, we will introduce and rationalize the implementation specifics to get specifications detailed enough for straightforward programming. We note that our presentation follows a waterfall-type fashion, but in practice our development process did not completely follow either the waterfall or the iterative model. Larman’s [48] and Fowler’s [49] books were found useful when documenting the software process and finding different presentation possibilities for software structure. The Unified Modeling Language (UML) was used whenever suitable, and techniques found in the aforementioned two books were applied selectively to find the best way of presentation.

4.1. Concepts

4.1.1. PnPAP

We first introduced the concept of PnPAP in [12]. PnPAP is a middleware architecture that provides easy-to-use communication facilities for applications. The idea is that application development is made substantially easier and faster by providing a high-level communications API with services spanning a wide range of application needs. Communication in P2P networks and suitability for mobile phones has been given special emphasis during the development, although there is no reason why the idea of PnPAP would not work with client-server protocols.
on desktop environments as well. Services provided by the API should include at least basic peer group management (create and join), search and download functionalities, as well as functionality to exchange context data between peers.

Figure 9 shows a rough architecture diagram of PnPAP. Inside, PnPAP features two essential engines for communication: the Protocol Engine and the Holistic Connectivity (HCon) Engine. Service requests from applications are forwarded to the Protocol Engine and, using intelligent control, one or multiple protocols are used to realize the requested service. Then, the protocol contacts HCon to send the first packet in the air. In many cases, HCon also has to choose which connectivity to use to provide service or if multiple ones should be used simultaneously. The Engines are differentiated from the components actually providing the service, and new capabilities - protocols, for instance - can be downloaded from other peers and taken into use on the fly.

Applications use the middleware through the Application Programmer’s Interface (API). For example, if Wellness wanted to download a file from a nearby peer, it would call a suitable function in the API. Then, using some logic, PnPAP would determine the optimal protocol to perform the download. Let us imagine PnPAP would select the JXTA protocol this time. Then, it would hook up the JXTA protocol instance to the optimal connectivity, say, Bluetooth. JXTA would thus connect to the other peer with a download request using Bluetooth and carry on the operation independently. Wellness would only need to listen for progress reports or error messages from PnPAP through the application callback interface and notify the user if needed.

![Figure 9. The architecture of PnPAP.](image-url)

Key factors that differentiate PnPAP from existing middleware systems include unprecedented compatibility and flexibility. By abstracting communication from
the application using the API, basically any communications protocol can be used to actualize services between peers. This offers superior possibilities: the used protocol can be determined by PnPAP at run-time by maximizing resource availability or minimizing protocol overhead, for instance. Even multiple protocols could be used simultaneously to maximize the amount of search results, e.g. when searching for files in a P2P network. This modularity provides for lightweight operation with applications not requiring rich functionality and versatility for those that do.

The superior features also extend into connectivities. As with protocols, communication is independent of connectivity from the application’s perspective. This enables run-time optimization of the networking technology as well. Therefore, an application built on PnPAP would only need to call one method in the API to support downloading a file over Bluetooth from a peer when the peer is close and to dynamically switch to using WLAN Internet connection if the peer moves out of range (requiring no further involvement by the application).

As we can see, the features of PnPAP are at least as convenient for the user as for the application developer. Until now, two users that need their devices to interoperate have themselves had the responsibility of ensuring that they use mutually compatible applications and hardware. Often, this has raised a need to install several applications that accomplish the same function simply due to protocol incompatibility (one protocol often being hardwired to a single application). The user has had to choose the application and connectivity himself before any communication could occur, and switching them has been laborious, manual work. In the end, however, protocols and connectivities are mere tools to transfer bytes. PnPAP enables their optimal integrated use by supporting an extendable set of protocols and connectivity support with minimal (or no) manual configuration.

In Howie et al. [50], this advanced feature set was given the name of Application Supernetworking. Application Supernetworking is a natural continuation of today’s mobile communication. Currently, interaction very rarely includes more than one media at once: calls have only voice, SMS messages only text. MMS messages are somewhere in the middle of 2G and 3G technologies. As we are moving towards using faster communication technologies and more capable end-user devices, the concept of Application Supernetworking can be fully implemented in the real world. Application Supernetworking contains at least three important elements [50]:

1. Multisessions and/or rich calls;
2. Plug-and-play interactions between sessions and applications;
3. Holistic Connectivity management.

Session is an abstract entity which is created when one process starts communicating with another. A session can utilize any connectivity such as WLAN or GPRS. Sessions can even be established between two processes running on the same device. Sessions are destroyed when communication is no longer needed. Multisession enables having multiple sessions open at any given time. The user might have, e.g. two video conference call sessions, one voice only call, and a remote desktop session active simultaneously. On the other hand, rich calls enable
multiple kinds of interaction over one session: one might have a video conference with audio, a file transfer and a shared desktop all running simultaneously over one call session. A true Supernetworking device can execute a number of processes using the same group of sessions in a multiplexed fashion, meaning application level session sharing and both multisession and rich call type of interaction. This is enabled by supporting plug-and-play interactions between sessions and applications, where received data is distributed to all the applications that need it, and sent bytes are gathered in a similar fashion. HCon management enables selecting the optimal connectivity available at a time. The selection may be based on highest bandwidth, lowest end-to-end delay or lowest price, for instance. The visibility of Application Supernetworking to the end-user can be kept to a minimum by implementing it as an independent, configurable middleware component. [51].

Furthermore, we want PnPAP to enable and encourage application cooperation. For example, we want the user to be able to easily find a list of possible interactions with a selected peer, no matter the application. These interactions would be provided either by the current application or by starting any of the other locally installed applications. A concrete usage scenario can be found in the use cases of Wellness in Appendix 1. We will take this feature into consideration in the analysis and design phases. More discussion has been presented in a publication [52].

Agile Content Push Control (ACPC) is another feature developed in the All-IP project. Its basic idea is to provide a seamless communication session startup regardless of whether the target peer has a suitable application installed to utilize the communication. If needed, ACPC downloads and installs the required application automatically. ACPC supports even local application cooperation by making the starting of yet uninstalled applications possible both locally and/or remotely. ACPC was presented by Kassinen et al. in [53] and further discussed in [54]. ACPC is out of the scope of this work, however.

4.1.2. Wellness Application

The Wellness application is an electrical, extended version of a regular sports notebook. It runs on a modern mobile phone and includes features for planning, monitoring and recording different types of physical exercises. For improved usefulness and usability, sensors are used to keep record of each exercise with detailed information ranging far beyond the capabilities of a regular heartrate meter. Based on sensor information and the selected plan for the workout, Wellness guides the user during her exercise. The application tells the user what type of exercise should be done next, and helps her in staying within the exercise specific target heart rates much like a state-of-the-art heart rate meter would.

Wellness makes use of peer groups: workout programs (i.e. plans) and results can be shared within a group of peers sharing the same interests. The application also enables following a peer’s workout in near real-time. Therefore, we will obviously develop Wellness on top of PnPAP. Potential users of the application can include ordinary sports hobbyists and enthusiasts as well as professional athletes, coaches and teams.

The basic concept of the Wellness application is shown in Figure 10. Interaction
concentrates around the peer group that can be a family or a football team, for example. Every peer joins in: some create new workout programs, and some download, modify and redistribute them. The results peers get by following the programs can be seen instantly by real-time monitoring and from stored result history. The idea is that the group motivates its members to work out effectively and safely, and to make it fun. One could gather tips for planning his future training not only from his own past results and programs, but also from the data of his peers.

Figure 10. Selected use cases of the Wellness application.

4.2. Analysis of Wellness

4.2.1. Use Cases

We will start off our analysis of the Wellness application by writing down the use cases. To quote Larman [48], "a use case is a collection of related success and failure scenarios that describe an actor using a system to support a goal". To put it another way, they are small stories depicting a chain of events producing some type of a meaningful end result to the user. The use cases for the Wellness application have been presented in Appendix I. We have presented the actors, pre- and post-conditions and description for each use case. Description provides an example of a successful scenario, whereas possible problems referenced in the text are given in the exceptions section for each case. For brevity, we have used a notably shorter format than the template by Alistair Cockburn suggested by Larman [48]. However, we argue that we still are able to catch the essence of the
cases.

To sum up the use cases, the Wellness application provides a few major features. Peer group management, workout program editing and sharing with search and download functionalities are included. Naturally, loading a program into the application to set up a workout must be supported also. One’s own and the peers’ workout history can be browsed with detailed information of each exercise. In addition to browsing the workout history, near real-time monitoring of ongoing workouts is possible.

4.2.2. Requirements

4.2.2.1. Documentation Style

Based on the application concept and with the help of the use cases, we defined a list of requirements for the Wellness application. The list is given in Appendix 2. The requirements were divided in three (more or less artificial) categories for added clarity: Environmental (ER), Functional (FR) and Usability Requirements (UR). ER aim to encompass the special requirements of the mobile implementation environment (‘where’), whereas FR strictly define what the application has to be able to do in the given environment (‘what’). UR specify how well the application should perform (‘how well’). In other words, UR are performance or Quality of Service (QoS) related requirements.

Each requirement is given a class defining its importance in the whole concept. The classes (MUST, SHOULD, MAY, etc.) have been used as defined by the Internet Engineering Task Force (IETF) Request For Comments (RFC) document 2119 [55]. Capital letters are used in the keywords to distinguish them from the ordinary words with less well-defined meanings. In short, MUST requirements are essential and have to be fulfilled to make for a successful implementation. They should also be tested properly after implementation to verify that they are met. SHOULD requirements are secondary, but they are recommended and there should be a good reason if one is broken. Class MAY is purely optional: they could also be called nice-to-have features, therefore they are the first ones to be left out of scope if, for example, development time runs short.

The requirements have been presented in a short form. Many of them, however, have more widespread consequences than it would first seem. For example, ER #1 "The application works on a standard S60 2nd Ed. FP2 mobile phone" needs further discussion. This single requirement has a profound effect on the design and implementation, as it defines a Series 60 mobile phone as the target environment. For example, the special characteristics of mobile devices and services discussed in chapter 2.5 have to be taken into account. On the other hand, most of the requirements are kept on a general level to leave room for more detailed decisions in the design phase.

4.2.2.2. Environment

Figure 11 shows an operational environment example for Wellness and PnPAP. The two S60 mobile phones are each connected to a FRWD sports computer via
Bluetooth. FRWD includes GPS, barometer and temperature sensors: heartrate can also be read from a separate heartrate transmitter belt. The mobile phone, FRWD and heartrate transmitter belt form the user's Body Area Network. FRWD produces workout related context information that is processed by PnPAP and the Wellness application. Using a wireless link, Wellness can download new workout programs from a peer group in the P2P network, as well as share its own programs and real-time context information. By choosing a suitable protocol, (with the help of PnPAP) Wellness can basically connect to any P2P network and any networked peer, including desktop and laptop computers and PDAs.

![Diagram](image)

Figure 11. The operating environment of PnPAP and the Wellness application.

We are targeting many types of sports, thus FR 3 "The application supports planning workouts for many types of sports". We should at least enable cross training and interval training in addition to basic training. The application should operate as well outside on a jogging course as inside at a gym. This also requires that the application functions in the absence of a fixed architecture, such as a UMTS network or a P2P hub connection. With pure P2P, this can be achieved. However, many commercial services are designed to include some form of administration which necessitates at least periodic connections to some fixed architecture. Pure P2P is difficult with mobile phones since the IPs phones typically get are private (behind NAT). Even if not specifically required by ER #1, a normal SIM card should do. However, at least TeliaSonera does provide public IP addresses for mobile phones when using a special access point.
4.2.2.3. Security

Sharing content within a closed peer group that requires some form of authentication to join can be an important use case because users may want to restrict sharing self-created content on their mobile phone to a specific set of people they already know. It must be assumed that it is unacceptable for material shared in a closed PG to end up in reach of someone not belonging to the PG unless a member specifically says so. On the other hand, the potential for open peer groups is raised by the observation of many people sharing (sometimes incredibly personal) material in a wide variety of blogs, personal web pages, web discussion forums and the like. These forums are known to be accessible by anyone. However, it is important to keep the user aware of the security level of her actions and to satisfy her expectations of security. When someone uploads a video clip to YouTube, it is expected to become publically available, thus, no security is needed. It is completely different if someone shares a file in his family’s closed PG.

The Wellness application may process valuable information. For example, if top-level athletes were using the system, the details of their training methods and current condition could be valuable information for their competitors. Also celebrities could be interesting targets. On the other hand, the training data of ordinary weight watchers would not be likely to inspire much effort on attacks to steal the data. Since we are targeting for a very limited prototype release, we have put limited effort on all aspects of security: data persistency, privacy, deniability, etc. Such requirements must be considered when developing the concept further.

4.2.2.4. Usability

As for Usability Requirements, the users want services to be easy to use through the whole process. Especially, setting up the application should be made as convenient as possible and we should avoid asking too many technical questions, automated setup and propositions should be preferred instead. In order to give maximum value to the user, the application should provide quick response times to queries and downloads. In addition, the user must be informed at all times about what the application is doing by using dialogs, progress bars, etc. Even a 10-second delay without any notification can produce a "the application died" type of feeling to the user.

4.3. Analysis of PnPAP

4.3.1. Use Cases

Next, we will start to scope the functionality we want PnPAP to be able to handle. This can be regarded as partitioning (or designing) the package Wellness and PnPAP form together. However, as we have chosen to present the development of PnPAP as its own entity, separately of Wellness, we need to analyze what is required of PnPAP to be able to develop it further. In this case, however, actors are not humans but applications. Only after designing at least the API for
PnPAP can we start designing the inner logic of Wellness, because at that point we will need to know what features need to be handled by the application itself. We started PnPAP analysis by transforming the use cases for Wellness to a more abstract level by replacing the word ‘workout program’ with a more abstract word ‘resource’, for instance. Then, we start seeing required functionalities on a more general level. At minimum, the Wellness application requires functionality to manage peer groups, to share, search and download resources and to exchange context information (in near real-time). Starting an interaction triggered by another application needs to be included, as well. These services would seem generic enough to be provided by PnPAP, as we can imagine the services to be useful also for other kinds of applications. No separate list of these use cases is shown in the Appendix for brevity, as the cases are included in the use cases 1, 4, 5, 9 and 11 of Wellness.

We limit the scope here and leave out context sensitivity, which we consider more intelligent processing of context information. For the purposes of the Wellness application, it is enough that we can collect information from sensors, store it locally and distribute it within the PG. Context sensitivity can be easily added into the system with the help of the dynamic state machines, for example. Context sensitivity has been widely studied elsewhere, for example in the CAPNET project at the University of Oulu [56].

4.3.2. Requirements

As the requirements for PnPAP are derived from the requirements of Wellness, we do not present a list for PnPAP separately. Instead, we will discuss the additional requirements related to PnPAP here. We design PnPAP from the beginning on to support the dynamic usage of protocols and connectivities, and we require ‘run-time expandability’, i.e. run-time possibility of downloading modules that provide new functionality. The middleware should also include intelligent optimization of these features. However, derived from the application requirements, the only connectivity we require is GPRS: others including Bluetooth and WLAN are optional. We claim that (at least in Finland) GPRS is ubiquitous enough to serve our purposes. We do not define the protocol(s) we will use yet. That is left as a design issue instead.

In short, PnPAP is required to provide features resembling a distributed P2P file system [3 pp. 323-366] with an easy-to-use interface for applications. However, to maintain a reasonable scope for our prototype we are required to ignore some requirements for distributed file systems given by Coulouris. Hardware and operating system heterogeneity must be supported. File replication should be supported to ensure data persistence, however that would also require means to control replica consistency. We will evaluate protocols in the design phase to find the best compromise between features and evaluated cost of implementation.

4.4. Design

This chapter presents the final design of the implemented software. In reality, especially the design of PnPAP experienced multiple iterations with added func-
tionality and refactoring of existing code. We consider the latest version the most optimal one where the worst teething problems have been fixed. Thus, we will not discuss changes that were made between the implementation iterations, unless we see we ended up in an avoidable common pitfall at some point. The diagrams that are referred to can be found in the Appendix.

### 4.4.1. Technologies

#### 4.4.1.1. Hardware

The Nokia Series 60 mobile platform on the Symbian operating system was already mentioned in the requirements. It was selected in the beginning of the All-IP project in 2004 as the platform because such devices are widely available and offer rich features with relatively good programming interfaces. Development of PnPAP was started on the Series 60 1st Edition platform with Nokia 7650 hardware. At the time of starting Wellness development in the spring of 2006, Nokia Series 60 3rd Edition phones with Symbian OS 9.1 were already available in stores. An example was the Nokia N91 that included the latest features such as WLAN connectivity. However, the 3rd Edition brought major changes to software development and porting our existing PnPAP code for S60 2nd Edition SDK supporting Feature Pack 2 [57] was quickly found too laborious for our limited resources. Thus, we ended up staying on 2nd Edition with FP2 that supports, e.g. Nokia 6680 and 6630.

The phones also needed SIM cards. We ended up going for a few SIM cards for the Octopus network, which is a development and testing environment for new mobile services [58]. Octopus provided us with GPRS data with static public IPs to allow for direct P2P connections between the mobile phones. Sonera SIMs could have also been used with their special access point that provides public IPs.

We also needed sensors to provide us with context data. We already had a few Bluetooth connected GPS modules. For the Wellness application we preferred to obtain more specific information of the workout, however. Heart rate was considered as the most important attribute, and we searched for devices that could be connected to an S60 phone to provide heart rate information in real-time. Not many suitable candidates were found, as most heart rate meters only included a non-standard interface to synchronize information with a PC. Few devices were found on the Internet that had a standard Bluetooth interface, but their cost and availability in Finland were uncertain.

Finally, we found FRWD Technologies based in Oulu, Finland. Their F-500 series includes a heartrate receiver compatible with Polar and Suunto heart rate transmitter belts, GPS receiver, barometer and thermometer. Using those sensors, the F-500 produces heart rate, location, distance, speed, altitude, temperature and air pressure information, which is more than we hoped for. The device can store the workout information in 1 second intervals in its internal memory, from which it can be downloaded to a standard PC by using Bluetooth. Then, the workout can be "re-experienced" with the Replayer software that FRWD provides. What is more important, the Bluetooth connection can also be used to stream the sensor information in real-time to, say, a mobile phone. As the F-500
has no display of its own, FRWD also has their own Mobile Player software for S60 phones to function as an external display with efficient information visualization. However, we are more interested in the ability to read the information to Wellness via PnPAP. We obtained a few F-500 packages with included heartrate transmitter belts.

4.4.1.2. Protocols and Connectivities

We aim to design PnPAP as protocol and connectivity independent and capable of adding such modules during runtime, but at this point we must fix the protocol and connectivity that we will implement first. After having completed the first functioning prototype, the amount of supported protocols and connectivities can be increased. However, we expect implementing them to be costly in resources and not to find ready-made implementations of either that would easily fit into the use of PnPAP. Thus, our resources may not allow for more than one module each. We will definitely use dynamic linking and polymorphic DLLs to implement the P2P protocols, such as DC++ or Napster, and connectivities like GPRS, to enable new protocols to be downloaded and taken into use at run-time.

We needed a P2P protocol that had an existing Symbian/S60 implementation that we could use without intellectual property right issues. Sadly, we did not find any. JXTA, for instance, did have example applications built on MIDP, but the implemented features did not seem very convincing in terms of being able to support our use cases. Therefore, we needed to implement the protocol ourselves, and we wanted a protocol that was simple (cheap to implement) but included the functionalities that we needed (and only them, as additional obligatory features would have only added unnecessary overhead). Documentation of the protocol specifics and existing implementations for desktop computers were naturally important attributes of our search.

DC++ was found to be the best candidate. It is a relatively simple text-based protocol that already had a wide open source community around it. We consider DC++ a first generation protocol, as a hub (indexing server) is needed and the hubs typically are not in contact with each other. There was no S60 implementation available, but a PersonalJava version of the DC++ client was available [59]. Also, documentation was close to perfect [60]. However, a downside of DC++ is that peer groups are not supported as such. The peers that have joined a DC++ hub cannot be further divided into different groups. As access to a hub can be restricted, the hub itself can be regarded as a peer group. Nevertheless, this way dynamic creation of peer groups cannot be supported. Creating a new hub requires a desktop computer and some manual configuration.

We also needed a protocol for exchanging context information. Some P2P protocols include presence support which could have been utilized, however DC++ did not. The P2P candidates that supported transferring real-time context were found too complex. Thus, we decided to rely on the industry standard, the Session Initiation Protocol (SIP). As such, it does not include means to exchange context information; however, its numerous extensions, including SIMPLE and PIDF, provide support for it. Nokia even provided a ready-to-use SIP plug-in for S60 with limited extension support.

In terms of connectivity, we wanted to connect to the Internet. As WLAN was
not supported by our hardware and using Bluetooth ad-hoc technology to find a gateway to the Internet was considered too complex for our purposes, we chose to implement a sockets based GPRS connectivity module. The S60 Software Development Kit (SDK) provided us with a sockets example that was a good foundation to build our connectivity on. Later, the same connectivity module was found to support other technologies as well. For historical reasons, however, we will refer to the module as "GPRS connectivity".

4.4.1.3. Programming Language

There was the question of whether using C++ or Java to implement our software. When starting the All-IP project, Java MIDP 1.0 and soon 2.0 were supported by Nokia platforms. Unfortunately, the standard MIDP without additional APIs is rather constrained. For example, it functions within its own sandbox, and access to the file system of the device is not provided. MIDP was targeted mostly at mobile games. Choosing Symbian C++ as our programming language was thus straightforward.

In the beginning, we also planned possibly implementing some applications with MIDP and designed PnPAP with a MIDP interface and provided common file system functionalities in the API. The MIDP interface was never implemented, and the file system functions were found redundant for C++ development due to Symbian’s own easily accessible functions. Thus, they are not presented in the following. We were ahead of our time, however: later, standard Nokia phones have started to support various extensions to the standard MIDP, including JSR 75 "FileConnection and PIM API", which provides the long-awaited access to the file system. Also, Python support was added in the mean time, which enables rapid prototyping and relatively wide interface support from the start. As we already had large amounts of source code written with C++, we stuck with it.

Symbian/S60 programming has quite a few special characteristics that the reader should be aware of. For example, in the following presentation, the function names end with 'L' if they are allowed to leave, which follows S60 and Symbian guides. The leave mechanism should be used when running out of resources. Thus, most functions also provide a return code to indicate success or failure. A 'C' prefix in a class name indicates the class is derived from the CBase class, and is intended to reserve heap memory. Derivation from CBase is not shown in the following diagrams because of this clear sign. An additional 'C' ending in a method name indicates an object is added to the cleanup stack. 2-phased construction is used for all objects with a 'C' prefix (objects reserving heap memory), but typically not shown in our diagrams for brevity - also methods are not shown in inheriting classes for the same reason. Standard Symbian practices of naming is followed throughout our presentation. Further details can be found for example in Tasker’s book [61] and in the Internet at NewLC [62].

4.4.1.4. Development Environment

Microsoft Windows XP, Microsoft Visual C++ .NET [63] with Nokia Carbide.vs 2.0.1 providing tools for Series 60 development were used in addition to the
suitable S60 SDK for the actual coding. The SDK includes an emulator that was used for preliminary testing as much as it could be. Some functionalities, Bluetooth, for example, was found difficult to test by using the emulator. Testing was conducted on true mobile phone hardware from the beginning in such cases. Bluetooth was found useful for repeated transferring of SIS application packages into our Nokia phones. Debugging was extremely difficult when running the software on the phones, however. Remote debugging would not work.

4.4.2. PnPAP API

We want PnPAP to be as easy to use as possible for the developer. With the PnPAP Application Programmer’s Interface, we thus aim at the highest abstraction level where functions are still atomic and practical. Through high abstraction, we also enable our dynamic protocol and connectivity control without effort required by the application. The API for PnPAP can therefore be derived in a quite straightforward fashion from its use cases. The interface is presented in Symbian C++ language in Appendix 3, divided into a few functionality categories. Only the relevant functions are listed.

If the function provides a result synchronously (the service has been performed by the time of return), the function is tagged with the keyword ‘Sync’. If the function returns a result via the callback interface, the keyword is ‘Async’ instead. A preliminary distinction between synchronous and asynchronous functions can be done based on whether the results of the function need to be formed using communication over the network interface. If they do, the function is best to return immediately and provide the results with a callback not to stall the application for the duration of the communication.

One can see that there is a slight difference in style between the ‘forward’ and ‘backward’ APIs, namely that the ‘forward’ functions all have a dedicated function, whereas many of their callbacks are integrated into the single Register-Return function. Both interfaces could have been implemented using either of the two techniques. However, our approach has two pros: firstly, the ‘forward’ API is as easy to use as possible, and secondly, the callback often stays the same even if new functionality is included. Therefore, applications using PnPAP need not be modified, even if typically their ‘engine’ part inherits the callback interface.

4.4.3. PnPAP Design

4.4.3.1. Client-Server Architecture

We wanted PnPAP to include Application Supernetworking support, i.e. to be able to optimize the use of sessions and connectivities across applications. Thus, we wanted the PnPAP "core" to be the same for all applications using it, making the optimization easier by directing traffic from all applications through a single entity. We found the Symbian client-server framework (CSF) suitable for this purpose. In contrast to what is typically regarded as client-server, CSF always functions within a single mobile phone. The client of CSF is a library with a well-specifed interface that handles the initialization, communication over and
cleanup of a server session and which application developers use to utilize the services of the server. The idea is that a user of the client does not need to know that the service is actually provided by a server. More information on the CSF can be found in Tasker’s book [61].

The client-side code of the CSF is typically statically linked to the application, whereas the server resides in its own executable. The server can either be started during system boot, or by the client upon a client application startup. The client(s) and the server operate in separate processes. This adds significant complexity to the communication between the two, as processes normally cannot access but their own memory space. Thus, exchanging information between the two is more difficult than inside one process.

Additionally, the client-server framework includes no built-in solution for asynchronous callbacks. Only synchronous and asynchronous calls from the client side are supported. The latter ones are typically implemented with shared request status objects. The client sends the server a request with possible input data and a request status object and then starts to wait until the server changes the state of the status object as finished. Then, the active object (derived from CActive) of the client is awakened and the client can begin consuming the processed data provided by the server.

4.4.3.2. PnPAP Classes

Figure A4.14 shows the class composition for the PnPAP client/server architecture. The application only sees the RPnpapSvrSession that implements the MPnPAPAppInterface. Typically, the "engine" of the client application owns and uses the server session object. Each server session has a dedicated CMainEngineController object running in the server process that forwards requests to the CPnpapServer object that produces the service. In the case of asynchronous services, the CPnpapReceiveHandler (derived from CActive) starts waiting for the server to finish. When it does finish, the CPnpapReceiveHandler calls an appropriate method in the MPnPAPCallbackInterface to notify the application and provide it with possible result data.

The previous sequence does not, however, support providing incoming text messages to the application, for example. To be able to provide data from the server to a client, the client needs to have an ongoing active request. Therefore, the client would need to have a 'dummy' request to the server at times when there is no other request. However, what if a text message arrived from a peer during a file download? Then, we would either have to wait for the download to complete to be able to show the message or to complete the download request, issue the 'dummy' request and then continue the download request. These options would either impact the usability negatively or make the design complex. Thus, in absence of other, workable solutions, we decided to handle these asynchronous cases with a file-based method. The CPnpapServer sends these asynchronous messages to an application-specific file in the file system, and the CPnpapAsynchMessageHandler is notified whenever the file changes. Then, the async handler calls the MPnPAPCallbackInterface just as the CPnpapReceiveHandler would have done, therefore the application does not know the difference. Of course, this file-based method requires all asynchronous data to be serialized to and deserialized from
This solution was, however, the best we could find that supports our multi-application scenario.

Figure A4.15 shows a more detailed class diagram of the PnPAP server side classes. The numerous subsystems all have different fill colors. CPnPapServer is the core of the server side. It owns all of the other subsystems and is responsible for coordinating operation. For example, the server creates suitable CConnectivities and hands out to protocols which can then start using them. Both the facade and observer patterns [48, pp. 461-471] are used in our model. In short, the facade pattern hides a subsystem behind an object - in our case, this is done with CConnectivity and CProtocol interface classes. The observer pattern is used in all of the subsystems. It enables the subsystems to be used very easily in another context, as the subsystem is unaware of the specific type of its observer (or listener). The subsystem only needs to know it implements the subsystem’s observer interface. The observer pattern is used in a high-grained fashion here, since events cannot be subscribed by event type - it is all or nothing.

CPnPapServer is a listener to all of the subsystems and implements all the listener interface methods. It holds true also for CConnectivity. However, the server is set as the listener only during the initialization phase of a CConnectivityGprs. If initialization succeeds, the observer for the CConnectivity is switched to whatever subsystem triggered the creation of the new connectivity - DC++, for example. The server still remains the owner of the connectivity. If initialization fails, the server can try initializing another connectivity. Here, CConnectivity-Bluetooth is used for Bluetooth communication with the FRWD F-500.

Lists of methods were drafted by defining the preliminary responsibilities for each subsystem. During implementation, they needed fine-tuning. The CHttp-Client was first designed to use CConnectivity for communication, but it was found that Symbian/S60 SDK includes a suitable HTTP library with example source code. The source code of the stack itself was not available to us, thus refactoring it to use our connectivity model was not possible. HTTP was used when testing dynamic protocol switching for the new protocol library download from a known server. There was also an existing SIP stack for S60 available; however, due to some problems with its early versions and the lack of CConnectivity support motivated us to implement our own simple SIP stack, discussed further in [54].

4.4.3.3. Context Data Model

We designed our context data model to support an extendable set of context information types. At this phase, we needed containers for altitude, heart rate, location, temperature, presence, time and speed. A container for a list of supported interactions was also included for use with Supernetworking.

4.4.3.4. GPRS Connectivity

Figure A4.16 shows the class diagram of the CConnectivityGprs class. Our implementation is based on the sockets example provided with S60 SDKs. Support for sending UDP and listening to TCP and UDP packets was added by us. The func-
tionality for UDP is separate from the functionality for TCP - thus, the features may be used simultaneously for one instance of GPRS connectivity class.

4.4.3.5. DC++ Protocol

The class diagram of the CProtocolDcpp class is depicted in Figure A4.17. The CProtocolDcpp class is the coordinator that is responsible for the hub traffic. In need, CProtocolDcpp creates a new CCllentConnection to handle either a download or an upload request originated locally or remotely. This architecture was already in use in the Mobile DC open source Direct Connect client implementation on PersonalJava. Mobile DC was found in that its source code was browsed when the protocol documentation at was unclear with a detail. Mobile DC was not ported, however, as we wanted to implement the library ourselves and notable differences between PersonalJava and Symbian C++ programming would have likely required some changes in any case.

4.4.3.6. Protocol and Connectivity Management

Until now, we have supposed that PnPAP can choose optimal protocols and connectivities. We plan on implementing the intelligence as dynamically configurable state machines. The state machine (SM) executor inside PnPAP would be able to load and start running a state machine after reading an SM definition from a local file. This way, we would enable the user and PnPAP to easily change the behavior of protocol and connectivity management altogether. Starting to prefer energy efficient and low overhead technologies could be done by adjusting the parameters of an already running state machine. A more detailed presentation of the State Machines and their utilization in PnPAP is given in and is out of the scope of this work.

4.4.3.7. PnPAP Intercommunication

In addition, to support Application Supernetworking in an optimal way, we need the PnPAP nodes to be able to negotiate about their preferences and select a common protocol-connectivity pair. This is done in a 'PnPAP back-end', called the PnPAP network, which is separate from the actual P2P networks used to transfer files, for example. We call the back-end traffic PnPAP intercommunication. It can also be used in exchanging context information between PnPAP peers if a suitable P2P protocol is not available to fulfill the task. Intercommunication is only partly within the scope of this work. More details can be found in Howie et al. and Harjula et al. The latter discusses using P2P SIP instead of regular SIP.

4.4.4. Wellness Application Design

Wellness is based on the Model-View-Controller (MVC) design pattern. The data (Model) is separated from the Controller object, which controls the actions
between the Model and the Views on the data.

4.4.4.1. Model

In addition to using the PnPAP context data model, Wellness includes its own model for representing workouts and their contents. We wanted the data model to support as many sports as possible. We also wanted to support sets and changing the workout program after it has already been started. The workout contains any number of sets, whereas a set can contain any number of exercises: An exercise could be "10 repetitions of bench press with 60 kg", or "30 s running at maximum effort", for example. The number of iterations for each set can be set separately. For instance, the single exercise "10 repetitions of bench press with 60 kg" could be set to be iterated for four times. For flexibility, a set can contain any number of exercises. Therefore, it is usable for a round of cross training at a gym, for instance. Each exercise in the workout can have its own heart rate and speed target limits, or they can be set equal by using the setters in the set of workout classes.

The exercise class supports targets of repetitions, time or distance. The description part tells the specific thing to repeat, for example, "rowing machine at medium effort". The class storing the result of an exercise, CWellnessExerciseResult, is derived from CWellnessExercise. It adds a result field (of the same type as the target type for the exercise) and statistics of the exercise such as heart rate minimum, maximum and average. Results are contained within the CWellnessWorkoutResult class which also specifies which exercise is currently active. There is no equivalent of a set for results, as we wanted the user to be able to perform the workout at any order she chooses, and thus the exercises are stored in the workout results list in the order they are input, regardless of the planned order.

4.4.4.2. Controller

The Controller of Wellness is the CWellnessAppUi class shown in A5.18. CWellnessAppUi has the ultimate control over everything - it owns and controls the views. It also owns the CWellnessEngine, which can be regarded as a subcontroller that handles the PnPAP interaction and owns the workout and context data received from and sent to PnPAP. The engine has a distributed callback framework, and different classes can be the observers of the engine at a given time. However, the CWellnessAppUi is the fixed observer for peer group related messages as well as ActivateInteractionL requests originating from PnPAP as the result of a supersession startup request.

4.4.4.3. View

States of the GUI are shown in A5.19. To summarize, the application can be managing peer groups, browsing for workout programs locally or remotely, editing local programs, looking at a list of peers currently in the joined peer group or
looking at any peer’s workout history. Many features end up in the training
views - the same ones are used whether the user herself is working out or if she
is watching her own or someone else’s workout in real-time or using the workout
history.

All the views are implemented with a view-container pattern commonly found
in Series 60 applications. The view classes inherit CAknView, whereas the con-
tainers inherit CCoeControl and MCoeControlObserver. The containers that in-
clude list boxes additionally inherit MEikListBoxObserver from the S60/Symbian
API. Furthermore, the list views inherit abstract CWellnessList<pview and CWell-
nessListContainer classes, that implement most of the functionality for such
views.

The training containers are based on the use of many CEikLabels and a CEik-
ListBox at the bottom of the screen to show the workout program and results.
They are updated with information received straight from the CWellnessEngine
through specific observer interfaces.

4.4.4. Graphical GUI Design

The layouts of the views were first sketched with paper and pen. Figures [12] and
[13] show final, actual screen captures of the Wellness application. Different types
of lists are used in many views to represent information, and actions are available
either in the Options menu or with the selection key, if a default action can be
safely determined. The training views are based on CEikLabels to show data.
In addition, there is a listbox on the bottom of each training view to show the
planned program and results.

The user interface was designed to be as flexible as possible for use in different
sports. We also wanted the user to be able to access as much information as
possible during the workout, but in a way that the most common information
would still be easily accessible. Our limited resources did not allow for more
elegant views that would have been more complex to implement. Thereby we
ended up using readily available GUI components. Graphical representation of
information certainly would have been a far better choice in many cases when
usability and utility are concerned.

Colors and sounds were utilized to get the user’s attention. For example, heart
rate or speed limit warnings were represented with blue and a low toned beep
when below the limit and with red and a high toned beep when above it, like in
typical heart rate meters. Also, reaching a set target in time or distance based
exercises notified the user with a beep and continuing the exercise showed the
target with red. As a fine detail, the user was able to add exercises into the
application during his workout to modify the program a little at run-time. The
quick workout feature was based entirely on this.

4.4.5. Deployment

Figure A6.20 shows a shared deployment/component diagram for both Wellness
and PnPAP. The S60 phone with Wellness and PnPAP is connected to the FRWD
F-500 via Bluetooth, and to other peers and the DC++ hub via GPRS. GPRS
is also used to register with the SIP server, which is not shown in the diagram as it only forwards messages from peer to peer. TCP/IP is used to connect to the DC++ hub, and TCP and UDP are utilized for connections with peers according to the DC protocol specification [60]. SIP messages are sent to and received from PnPAP peers on UDP.

### 4.4.6. Sequence Diagrams

The sequence diagrams for Wellness use cases are shown in the Appendix 6 starting from Figure [A6.21]. No separate sequence diagrams are presented for PnPAP, as the use cases for PnPAP are a subset of the ones for Wellness. Furthermore, sequences are not shown for use cases that contain only local actions, which makes them trivial to implement.

In [A6.22] the list of shared files is not sent to the DC++ hub. Instead, the hub only gets notified of the amount of shared bytes. DC protocol specifies that...
the hub does not know what files those shared bytes actually consist of, but search requests are forwarded to the peers via the hub, instead. The provided sequences should be self-explanatory. Local supernetworking depicted in use case 11 is presented in more detail in a publication of ours [52].
5. RESULTS

This chapter discusses the outcome of our software development based on the specifications presented earlier. The prototypes are studied both quantitatively and qualitatively. Quantitative investigation includes measurement results of the run-time features and performance of the software. Qualitative information is produced with end-user evaluation that was performed to find out the reception of and potential for the Wellness application.

5.1. Compliance

We used both the use cases and the requirements to test if our prototypes comply with the specifications. As all the use cases of PnPAP were derived from the use cases of Wellness, we only needed to test the latter. Table 1 shows the results of the use case tests.

<table>
<thead>
<tr>
<th>Use case #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test result</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Accept</td>
<td>Pass</td>
<td>Fail</td>
<td>Not tested</td>
<td></td>
</tr>
</tbody>
</table>

The functionality for Use case 1 was restricted in our prototype, because we wanted to ensure that the users join the same peer group. This was necessitated by the low number of F-500 hardware we had available. We wanted to maximize the amount of accessible content and interaction between the users during the evaluation and, for that reason, we did not want our users to create their own peer groups. Creating peer groups was hindered also since setting up a DC++ hub is relatively complicated. Only joining the default peer group, i.e., our DC++ hub, was enabled. To further maximize the quantity of shared information, the functionality of use cases 4 and 8 was automatic.

Our tests showed that the functionalities for Use cases 1-8 worked. Some test runs for Use case 9 showed anomaly. It appeared that, for an unknown reason, Wellness crashed during some runs when testing use case 9. We had to ignore the randomly occurring problem, however - for debugging it was difficult and our schedule was tight. This was caused by the fact that the implementation process was considerably behind its schedule. The test result was given the status of 'accept' to distinguish it from the ones that had no issues during testing. Use case 10 worked as expected. However, use case 11 produced the first total test failure. With the resources we had available, we could not get the supersession functionality to operate correctly. Trying to begin following a peer’s workout using the NaviP2P application did start Wellness, but we could not achieve Wellness to switch to the correct view and begin receiving context data. After trying to fix the functionality, Wellness started crashing. These problems were also due to our tight schedule and the difficulty of on-device debugging. Use case 12 was not implemented nor tested since it was optional and our resources did not allow for it.
5.2. Benefits of PnPAP

5.2.1. Benefits in Software Development

To provide data on the additional overhead introduced by the PnPAP architecture, we performed some measurements on the prototype. Table 2 shows the number of source lines of code (SLOC) for components of our PnPAP architecture, including the Wellness application. Only the code in cpp source files is taken into account, without headers. Sizes in bytes for each component are also given. The table shows that the application and the PnPAP Engine are almost equal in the amount of SLOC. In a non-PnPAP application, the DC++ protocol and GPRS connectivity would have been implemented inside the application. By providing them with middleware, we achieve many advantages. First, persistent memory capacity is saved, and second, the same protocol and connectivity code does not exist in multiple application executables therefore wasting space. Also, run-time memory consumption can be lowered for the same reason.

Furthermore, we assume development time of an application is lowered with the help of existing services that developers can utilize. Savings are achieved in nearly all phases of software development from design to implementation and testing. Lack of comparable statistical information regarding required software development resources in cases with and without PnPAP prevents us from presenting more accurate information, however.

We have implemented many other applications on our PnPAP prototype. The application prototypes include the File Sharing application, whose name describes its use very well, and NaviP2P [66], [67], a navigation application that utilizes context information sharing in peer groups to show peers on a map. We have also implemented the Real-time application, which provides communication between peers utilizing peer-to-peer media streams. All these applications were implemented on PnPAP with only little or no modifications on the API or underlying architecture. This proves that our approach is easily extendable and can serve a wide variety of networked mobile applications.

One possible way of estimating the savings that an application developer gets by using PnPAP is to consider only the amount of SLOC: if in total 62% of the SLOC are in PnPAP, only 38% of the SLOC needs to be implemented by application developers, thus reducing the amount of required resources by more than half. On the other hand, PnPAP includes program code that probably would not appear in a non-PnPAP application. If only the protocol and connectivity SLOC were taken into account, application developers would need to implement about 20% less program code, thereby decreasing the amount of required resources by almost a fifth. The estimated benefit can therefore be anywhere between 20% and 60%.

<table>
<thead>
<tr>
<th>Component</th>
<th>SLOC</th>
<th>Proportion</th>
<th>Size [bytes]</th>
<th>Proportion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wellness</td>
<td>9 851</td>
<td>38%</td>
<td>80 416</td>
<td>32%</td>
</tr>
<tr>
<td>PnPAP Engine</td>
<td>10 841</td>
<td>42%</td>
<td>112 144</td>
<td>45%</td>
</tr>
<tr>
<td>DC++ Protocol</td>
<td>3 827</td>
<td>15%</td>
<td>44 852</td>
<td>18%</td>
</tr>
<tr>
<td>GPRS Connectivity</td>
<td>1 310</td>
<td>5%</td>
<td>12 676</td>
<td>5%</td>
</tr>
</tbody>
</table>
Table 2 also shows that the sizes of PnPAP components remain adequately small in order to be able to download and install them on the fly, even on current mobile devices. End-user observable latencies related to dynamic protocol switch were measured on a Nokia 6600 phone using GPRS and are given in Table 3. The delay data was gathered using a set of one hundred measurements.

<table>
<thead>
<tr>
<th>Delay Component</th>
<th>Average Delay</th>
<th>Min Delay</th>
<th>Max Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downloading DC++ Protocol DLL</td>
<td>10.774 s</td>
<td>9.891 s</td>
<td>19.922 s</td>
</tr>
<tr>
<td>Load DC++ DLL into RAM</td>
<td>169 ms</td>
<td>156 ms</td>
<td>188 ms</td>
</tr>
<tr>
<td>Function Call to DC++ DLL</td>
<td>&lt; 1 µs</td>
<td>&lt; 1 µs</td>
<td>&lt; 1 µs</td>
</tr>
<tr>
<td>Function Call inside DLL</td>
<td>&lt; 1 µs</td>
<td>&lt; 1 µs</td>
<td>&lt; 1 µs</td>
</tr>
</tbody>
</table>

One additional delay component, in comparison to a non-PnPAP approach, includes downloading the polymorphic Dynamic Link Library (DLL). We must note, however, that this approach makes the application package smaller, thereby making its download time shorter. DLLs only need to be downloaded once, unless they are removed or updated. Added delay is also caused by taking the DLL into use (i.e. loading it into RAM and doing some initialization). This delay is evident each time the DLL is used. We also measured if function calls to the protocol DLL would produce greater overhead than local function calls inside a DLL, e.g. an application. Results indicate that using a loaded DC++ DLL is practically no different in delay sense than using a hard-coded protocol. We are thereby lead to formulas for the additional delay presented by our approach:

1. \( t_{total} = t_{download} + t_{load} \)
   when the protocol DLL is not already locally available, and

2. \( t_{total} = t_{load} \)
   for the following instances of use.

In practical applications, basic protocol and connectivity DLLs should be included in the installation package. During development, we created a PnPAP SIS installer file that contained the Dummy and DC++ protocols and GPRS connectivity libraries in addition to both PnPAP client and server executables. We then attached the PnPAP SIS file into application SIS installer files thereby making the distribution of a complete package easier. This obviously increased the size of the application SIS significantly. However, we believe mobile applications are often installed using a fixed (USB) or a short-distance wireless connection (Bluetooth) to a desktop computer, and an increase in the amount of transferred data has a smaller effect using those high-bandwidth connectivities than GPRS, for example. Thus, considering that modern mobile terminals often have plenty of persistent storage space, the most common protocol and connectivity libraries should be included in the installer file.

Run-time downloading of libraries should only be used in the need of new (or updated) functionality. If a new (version of a) library was released, it could even be downloaded automatically when the terminal was otherwise idle. Doing this would maximize the number of situations where formula 2 applies, thereby minimizing the delay that the user can observe when performing a function.
This minimization would be done, however, at the cost of possibly installing libraries that will never be used, thereby adding unnecessary communication and persistent storage consumption.

5.2.2. Dynamic Protocol and Connectivity Management

To test whether the dynamic protocol switch feature works in practice, we setup a specific test case to try it. We used File Sharing, another prototype application built on PnPAP, to search for files with specific keywords. We made sure that the only installed protocol did not find any results. Next, PnPAP informed the user via File Sharing that no results were found and that another protocol would be downloaded to perform an added search. The DC++ protocol library was then downloaded via HTTP from a known server and the additional search using the new protocol was begun after the completion of the download. Then, DC++ successfully returned a few search results that were shown in the search results view of File Sharing. The resulting files could be then downloaded on the user’s demand.

The test was successful and demonstrated the feasibility of our approach. In this worst-case scenario, PnPAP needed to download DC++ which inflicted significant additional delay on the search operation on top of the lag caused by the two operations being sequential. However, we claim the end-user still benefited from the dynamic protocol switch.

In a non-PnPAP case with no results from the first round of search, the end-user of a file sharing application would have needed to try searching again with slightly modified keywords. This approach could be repeated for many times, each adding to the total delay of finding results. It is possible that no results would ever be found, because the peers within the reach of the application (and protocol) contained no suitable ones. The user would then either have to give up or try searching again using another file sharing application that used a different protocol. Changing over from using one application to another can be an arduous task that possibly requires finding, downloading and installing a new application.

PnPAP provides an expandable set of protocols for applications, thereby enabling end-users to perform searches efficiently - from peers supporting a heterogeneous set of protocols - using a single application. Optimal connectivity management and performing multi-protocol searches concurrently instead of sequentially would make searches even more effective.

5.3. End-User Evaluation

We wanted to find whether the novel Wellness concept has potential and what typical end-users think about the prototypes. We also wanted to know if Wellness fulfilled its goal of motivating people to take regular exercise and the features that were found especially useful. To gather as unbiased information as possible, we measured usability and opinions with an end-user evaluation. The evaluation was conducted in collaboration with two other Master’s Thesis writers, who had their own set of questions in the shared questionnaire, excluding a few background information related questions that were shared. The first part of the questionnaire
was about the NaviP2P application, and it was developed and the results were discussed by Ohtonen [67]. The last part by Goman [68] concentrated on studying the market potential of both NaviP2P and Wellness as well as mobile P2P in general.

We needed a set of equipment for each end-user, as we assumed the users did not own a suitable FRWD sports computer already. A Nokia Series 60 based mobile phone (either 6630 or 6680) was loaned to each user with a FRWD F-500 sports computer package including a heart rate transmitter belt. We installed PnPAP and Wellness in the phone before giving it to a user. We only had three F-500 packages which limited the size of each group of end-users to equal number of persons. To increase the number of respondents to improve the reliability of the data, we arranged the evaluation for five separate groups. All the users were unbiased and volunteered for the evaluation. Many groups had people who were already familiar with each other. Each group had the equipment for approximately one week after which the next group started their evaluation. The evaluation took place from July to October 2006.

The evaluation consisted of two parts. The first one was an opening meeting where the background and motivation for the evaluation were presented to the users. NaviP2P and Wellness were introduced to the users with the help of a short Powerpoint slide show. No further guidance on the use of the applications was given before asking the users to take their questionnaire sheets and reply to the background questions. Next, they were asked to carry out a few basic tasks with the applications, first with NaviP2P and then Wellness. After performing a task they answered related questions by filling their answers on the sheets (Appendix 9), one sheet for each application. The users were allowed to ask questions during this phase. After they had finished the tasks, they were allowed to leave the meeting with the equipment and the questionnaire form which began the second part of the evaluation.

The second part involved free use of the application. The users were given free tickets to a well-equipped gym to allow them to try the application during gym workouts or gymnastic exercises as well. The respondents were asked to try the applications for a few times during the week and then fill in the rest of the questionnaire including Goman’s part. There was again one sheet for each application. The sheet for Wellness is presented in Appendix 9. The groups then returned the anonymous forms and the equipment. During the evaluation, a group in the SmartGroups WWW service (now closed down) was provided to function as an approximation of a focus group [69, pp. 214-217]. The group was intended for free discussion regarding the applications and possible problems, for example. However, the group received only few messages that did not provide any valuable added information.

The questions for Wellness encompassed many components of system acceptability defined by Nielsen [69, p. 25]. The first part included mainly usability related questions with score from 1-4 (1=disagree, 4=agree), more specifically "easy to learn" and "efficient to use" related questions. The second part first asked the respondent to evaluate how many times he or she had used the application and its features. Next, there were questions about usability and utility as well as cost and reliability to determine practical acceptability. The score scale was the same score as in part 1. In addition, there were open questions in both parts. Lastly, the respondent was asked to give the application a grade from 1 to
5.3.1. Results

The results of the evaluation are shown in Appendix 10. Only the results for closed questions are included. The results were processed anonymously, but we did tag each evaluation form with the group’s id. All the parts of the form were stapled to maintain a link between the background information and all the response sheets. Based on the group identifications, an average was calculated over the respondents of each group. Constant increase or decrease in the responses could have meant differences in the realization of the opening meeting between the groups or differences in how the related hardware and software worked. However, no such constant change was discovered which meant conditions of uniform quality for all groups.

5.3.1.1. Respondents

There were 15 respondents in total, 3 female and 12 male. Their ages ranged from 21 to 40. Everyone was highly educated or at least was studying at an institution of higher education. Everybody enjoyed working out: half of them exercised at least twice a week, and the other half three times a week or more. Their sports included walking with and without sticks, jogging, running, roller skating, gymnastics, gym workouts, cycling, swimming, combat sports, golf, floorball, football, orienteering, skiing and badminton. The source of motivation behind their exercise ranged from personal health and well-being to fitness, weight control. Some worked out just for fun, whereas two of them were interested in competing.

We will first discuss the results on average. Using statistics to analyze the results, we saw differences in the answers between users that had used a heart rate meter before (8 respondents) and those that had not (7 respondents). We split the respondents into two different groups based on this fact for another analysis.

5.3.1.2. Average over all responses

Table 4 shows the average calculated over the answers of all respondents. On average, the respondents had used smartphones quite often, even more often than P2P applications. After the basic tasks, the users seemed to appreciate the user interface as the averages lean towards 'easy' and 'fast' for each use case.

After one week of testing, the opinions were not as positive. First of all, the application had not been used very much. There were on average less than 3 instances of use during the whole week. Editing workout programs and working out were the top-rated features, whereas the peer group related features did not seem interesting. Downloading programs and examining peers’ workouts in real-time or afterwards were used rarely. Browsing one’s own workout history was somewhere in between these two extremes in popularity.

Even if the users did not have to pay for the use of the prototypes during the evaluation, the respondents thought the estimated total costs for using the
service exceeded the benefits. It also seemed that Wellness failed in its goal of motivating people to exercise more - hardly anyone admitted working out more during the evaluation. It must be noted, however, that all the participants already worked out quite intensively and there was little room for more. Most of the people thought Wellness was no or only little better than their previous method of recording workout results. The devices were found relatively suitable for use in such an application, however. Easiness of use was regarded worse than during the opening meeting. Perhaps the features included in the basic tasks were considered easy enough, but the remaining features were not. User satisfaction was not very high, and few would like to continue using Wellness in the future. A positive thing was that the users were not too concerned over sharing their context information in the peer group, which is a profound requirement for the community features in these type of applications. The grade was not very high, only 2.29 out of 5.

Table 4. Results of end-user evaluation on average.

<table>
<thead>
<tr>
<th>Question</th>
<th>Average</th>
<th>S.d.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Used smartphone often</td>
<td>3.20</td>
<td>1.08</td>
</tr>
<tr>
<td>Used P2P applications often</td>
<td>2.14</td>
<td>1.10</td>
</tr>
<tr>
<td>Used mobile Internet often</td>
<td>2.47</td>
<td>0.99</td>
</tr>
<tr>
<td>Installed smartphone apps often</td>
<td>1.73</td>
<td>0.70</td>
</tr>
<tr>
<td>Number of smartphone apps bought</td>
<td>0.40</td>
<td>0.91</td>
</tr>
<tr>
<td>Opening meeting</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Login was easy</td>
<td>3.13</td>
<td>1.13</td>
</tr>
<tr>
<td>Login was fast</td>
<td>3.33</td>
<td>0.98</td>
</tr>
<tr>
<td>Searching was easy</td>
<td>3.14</td>
<td>1.03</td>
</tr>
<tr>
<td>Searching was fast</td>
<td>3.00</td>
<td>1.04</td>
</tr>
<tr>
<td>Starting a workout was easy</td>
<td>3.43</td>
<td>0.65</td>
</tr>
<tr>
<td>Starting to follow workout was easy</td>
<td>2.90</td>
<td>0.99</td>
</tr>
<tr>
<td>Starting to follow workout was fast</td>
<td>2.90</td>
<td>0.99</td>
</tr>
<tr>
<td>After free use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total amount of service used (hours)</td>
<td>2.03</td>
<td>1.20</td>
</tr>
<tr>
<td>Total instances of use (times)</td>
<td>2.80</td>
<td>1.26</td>
</tr>
<tr>
<td>Creating/editing program (times)</td>
<td>2.36</td>
<td>1.22</td>
</tr>
<tr>
<td>Downloading program (times)</td>
<td>0.64</td>
<td>0.63</td>
</tr>
<tr>
<td>Working out using the service (times)</td>
<td>2.21</td>
<td>1.12</td>
</tr>
<tr>
<td>Examining own workout history (times)</td>
<td>1.21</td>
<td>1.05</td>
</tr>
<tr>
<td>Following peer’s workout (times)</td>
<td>0.29</td>
<td>0.61</td>
</tr>
<tr>
<td>Examining peer’s workout history (times)</td>
<td>0.57</td>
<td>1.16</td>
</tr>
<tr>
<td>Total costs vs. benefit reasonable</td>
<td>1.60</td>
<td>0.83</td>
</tr>
<tr>
<td>Worked out more than usual</td>
<td>1.53</td>
<td>0.52</td>
</tr>
<tr>
<td>Would continue to work out more</td>
<td>1.53</td>
<td>0.64</td>
</tr>
<tr>
<td>Service was better than previous method</td>
<td>1.83</td>
<td>0.72</td>
</tr>
<tr>
<td>Device was suitable for service</td>
<td>2.47</td>
<td>0.83</td>
</tr>
<tr>
<td>Using the service was easy</td>
<td>2.40</td>
<td>0.74</td>
</tr>
<tr>
<td>I am satisfied with the service</td>
<td>2.07</td>
<td>0.59</td>
</tr>
<tr>
<td>Sharing my data aroused concern</td>
<td>1.47</td>
<td>0.64</td>
</tr>
<tr>
<td>I would like to continue using the service</td>
<td>1.93</td>
<td>0.62</td>
</tr>
<tr>
<td>Grade 1-5</td>
<td>2.29</td>
<td>0.91</td>
</tr>
</tbody>
</table>
5.3.1.3. Split based on heart rate meter use

Table 5 shows the statistics of responses grouped based on whether the respondent had used a heart rate meter in his/her exercise before. Our hypothesis is that the people that have already used heart rate meters should be more interested in Wellness than the ones that have not. Using a heart rate meter shows that the user wants to work out efficiently and with discipline. Wellness should match these goals very well, whereas it may be of no use for people who have no interest in monitoring their heart rate during exercise or planning their workout beforehand, for example.

In the opening meeting these two groups had almost equal responses. After free use, however, there is a significant 1-hour difference in the duration of using the service. It would appear it was spent for working out with the application enabled and examining one’s own workout history. Even if the heart rate meter users thought the costs of using Wellness were somewhat unreasonable, even more so than the non-HR meter users, they replied they worked out more and would continue to work out more than their meterless friends. The HR meter users also responded that Wellness was better than their old method, which would indicate that they did in fact see value in the added services Wellness included. Meter-equipped respondents also considered Wellness easier to use than the other group. It is possible that the meterless users are not interested in the information a HR meter or Wellness can provide, which is why they regard using it more difficult. It may also be that they lack experience to compare Wellness with. The HR meter users also were more willing to continue using Wellness and the grade they gave is significantly higher. Even the score they gave leans towards the negative on average, however.

5.3.1.4. Open questions

There were a few open questions in the response sheet. We will first discuss what sports the respondents thought Wellness is especially suitable for. Some replied the application suits only outdoor sports well: walking with and without sticks, jogging, orienteering, skiing, cycling and roller skating, for example. One respondent said Wellness is suitable for individual sports but that it helps in supportive exercises in other sports as well. Some users thought the application suits only professionals, whereas others replied that it is suitable for all levels. So far, the users agreed with us. We designed Wellness to be as generic as possible to suit as many different sports as possible. We anticipated that having to buy the required hardware and carry the equipment would not appeal to everybody. As the question was open, some respondents probably considered only those sports that they were interested in. We can conclude that Wellness is suitable for many kinds of sports both outdoors and indoors and at different levels.

Miscellaneous comments collected from the rest of the open questions revealed opinions that can explain some of the results in the closed sections. Many considered Wellness a fine idea, but found fault with flaws of the implementation. Everybody thought Wellness was too unstable, which was also detected during our compliance testing. It was unfortunate that we were not able to solve the is-
Table 5. Results of end-user evaluation with grouping based on previous heart rate meter use.

<table>
<thead>
<tr>
<th></th>
<th>Used HR Average</th>
<th>Used HR S.d.</th>
<th>Not used HR Average</th>
<th>Not used HR S.d.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Background</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Used smartphone often</td>
<td>3.00</td>
<td>1.07</td>
<td>3.43</td>
<td>1.13</td>
</tr>
<tr>
<td>Used P2P applications often</td>
<td>1.57</td>
<td>0.53</td>
<td>2.71</td>
<td>1.25</td>
</tr>
<tr>
<td>Used mobile Internet often</td>
<td>2.75</td>
<td>1.04</td>
<td>2.14</td>
<td>0.90</td>
</tr>
<tr>
<td>Installed smartphone apps often</td>
<td>1.75</td>
<td>0.71</td>
<td>1.71</td>
<td>0.76</td>
</tr>
<tr>
<td>Number of smartphone apps bought</td>
<td>0.75</td>
<td>1.16</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td><strong>Opening meeting</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Login was easy</td>
<td>3.25</td>
<td>1.16</td>
<td>3.00</td>
<td>1.15</td>
</tr>
<tr>
<td>Login was fast</td>
<td>3.25</td>
<td>1.16</td>
<td>3.43</td>
<td>0.79</td>
</tr>
<tr>
<td>Searching was easy</td>
<td>3.00</td>
<td>1.20</td>
<td>3.33</td>
<td>0.82</td>
</tr>
<tr>
<td>Searching was fast</td>
<td>2.75</td>
<td>1.16</td>
<td>3.33</td>
<td>0.82</td>
</tr>
<tr>
<td>Starting a workout was easy</td>
<td>3.29</td>
<td>0.76</td>
<td>3.57</td>
<td>0.53</td>
</tr>
<tr>
<td>Starting to follow workout was easy</td>
<td>3.00</td>
<td>0.89</td>
<td>2.75</td>
<td>1.26</td>
</tr>
<tr>
<td>Starting to follow workout was fast</td>
<td>3.00</td>
<td>0.89</td>
<td>2.75</td>
<td>1.26</td>
</tr>
<tr>
<td><strong>After free use</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total amount of service used (hours)</td>
<td>2.50</td>
<td>1.41</td>
<td>1.50</td>
<td>0.65</td>
</tr>
<tr>
<td>Total instances of use (times)</td>
<td>2.88</td>
<td>1.36</td>
<td>2.71</td>
<td>1.25</td>
</tr>
<tr>
<td>Creating/editing program (times)</td>
<td>2.13</td>
<td>1.25</td>
<td>2.67</td>
<td>1.21</td>
</tr>
<tr>
<td>Downloading program (times)</td>
<td>0.63</td>
<td>0.74</td>
<td>0.67</td>
<td>0.52</td>
</tr>
<tr>
<td>Working out using the service (times)</td>
<td>2.50</td>
<td>1.31</td>
<td>1.83</td>
<td>0.75</td>
</tr>
<tr>
<td>Examining own workout history (times)</td>
<td>1.00</td>
<td>1.07</td>
<td>1.50</td>
<td>1.05</td>
</tr>
<tr>
<td>Following peer’s workout (times)</td>
<td>0.13</td>
<td>0.35</td>
<td>0.50</td>
<td>0.84</td>
</tr>
<tr>
<td>Examining peer’s workout history (times)</td>
<td>0.63</td>
<td>1.41</td>
<td>0.50</td>
<td>0.84</td>
</tr>
<tr>
<td>Total costs vs. benefit reasonable</td>
<td>1.50</td>
<td>0.53</td>
<td>1.71</td>
<td>1.11</td>
</tr>
<tr>
<td>Worked out more than usual</td>
<td>1.75</td>
<td>0.46</td>
<td>1.29</td>
<td>0.49</td>
</tr>
<tr>
<td>Would continue to work out more</td>
<td>1.75</td>
<td>0.46</td>
<td>1.29</td>
<td>0.76</td>
</tr>
<tr>
<td>Service was better than previous method</td>
<td>2.00</td>
<td>0.76</td>
<td>1.50</td>
<td>0.58</td>
</tr>
<tr>
<td>Device was suitable for service</td>
<td>2.50</td>
<td>0.53</td>
<td>2.43</td>
<td>1.13</td>
</tr>
<tr>
<td>Using the service was easy</td>
<td>2.63</td>
<td>0.52</td>
<td>2.14</td>
<td>0.90</td>
</tr>
<tr>
<td>I am satisfied with the service</td>
<td>2.00</td>
<td>0.53</td>
<td>2.14</td>
<td>0.69</td>
</tr>
<tr>
<td>Sharing my data aroused concern</td>
<td>1.50</td>
<td>0.53</td>
<td>1.43</td>
<td>0.79</td>
</tr>
<tr>
<td>I would like to continue using the service</td>
<td>2.14</td>
<td>0.69</td>
<td>1.71</td>
<td>0.49</td>
</tr>
<tr>
<td><strong>Grade 1-5</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.62</td>
<td>0.92</td>
<td>1.83</td>
<td>0.75</td>
</tr>
</tbody>
</table>

Sues related to the software implementation in time. We were already significantly behind our end-user testing schedule, and we had to begin the tests regardless of the knowledge of annoying faults. Thus, the users had to tolerate them. The users’ loss of patience due to the flaws may have had a significant effect on their answers, but it is difficult to estimate the extent of the effect afterwards.

Many users wanted better instructions. This is in no way surprising, because the users did not get any guidance on how to use the application after the opening meeting. No printed or electronic manuals were available even in the application itself. This is probably another significant factor behind the results of the closed questions. Instructions would have likely increased the score, because especially for the users not already familiar with S60 user interface it must have been difficult to try navigating through the Wellness GUI. Furthermore, the users were native
Finns, but the user interface was in English. Some would have hoped having a Finnish version of Wellness instead.

Lack of instructions can be attributed to the fact that Wellness was only an early prototype. However, this must have played its part in the users’ evaluation. The user interface also received some criticism, which can at least partially be attributed to the same thing. Especially editing workout programs was considered laborious on the mobile phone. The training views were also considered too complex to use during an exercise. Some also considered all the equipment difficult to carry during exercise. Nonetheless, added features were hoped for - presentation of statistics in graphical form, for example. Many wished for more workout programs and templates to start with.

5.3.1.5. Summary

The instability of Wellness stood out from the rest of the users’ comments. It was clearly the greatest issue related to the application. It was unfortunate that we were not able to fix the worst bugs before the evaluation. Now, it is difficult to tell if the negative responses about the application were in fact due to the instability or by something else.

Many thought the user interface of Wellness was not perfect, and some respondents also found it difficult to carry all the required equipment with them during their exercises. Low scores on the usability of the application can at least partly be attributed to the lack of instructions. It seems the opening meeting was not sufficient in teaching the users how to use the system. Nonetheless, the users did at least manage to edit programs and work out, which were the most popular features.

For some reason, the peer group related features were not given very much attention. This can be due to the very limited size of the peer group together with the fact that PnPAP had a flaw that required the F-500 device to be turned on if the user was to join the peer group. The F-500 ran on batteries, and it is likely that the users turned them off whenever not working out. Thus, the users were not joined in the peer group for most of the time, and the responses possibly reflect the inability to find peers to interact with.

We could not find evidence for our ambition to motivate users to exercise. It is probable that the instability affected these answers as well. The two respondents that were interested in competitive sports used the planning and monitoring features significantly more than the others. On the other hand, the two were less eager to download programs from their peers and to interact with them altogether using the application. It is possible that they wanted to focus on themselves and did not expect to find any useful information from their peers. However, a peer group of two with a coach or a personal trainer could be more feasible for professionals. The competitors were the only ones who did not find the total costs of the service greater than the benefit.

For some reason, the respondents that had themselves bought at least one smart phone application responded to many questions more positively than the others. They thought Wellness was easy to use during the opening meeting, but the situation changed during the free use, however. Nevertheless, they had used almost all features more than the other respondents. They were also more willing
to continue using Wellness after the evaluation and gave an average grade of 3, which was the highest average grade from any studied subgroup. Their more positive attitude towards Wellness may be explained by the quality of mobile applications in general: if the users were accustomed to poorly functioning mobile software in general, they were likely to feel less upset by the deficiencies of Wellness.

The last two groupings were not studied more widely, as the number of respondents was considered too imbalanced in the two cases. The level of statistical reliability was not very high to begin with, because of the limited number of users in our evaluation. Studying groups of two to three persons was not considered very safe. Regardless, we wanted to find interesting antitheses within the statistical data and these two groupings were found with notable differences.

All in all, we did not get the responses we were hoping for. However, the idea behind Wellness was highly appreciated. Probably in other circumstances the results could be very different, if the software was tested more properly for bugs and usability issues before beginning the actual evaluation.
6. DISCUSSION

In chapter 2, we discussed the state-of-the-art of mobile P2P and equipment designed to help taking physical exercise including the major technological enablers to establish such services. To our knowledge, concepts similar to PnPAP and Wellness did not already exist. Therefore, comparing our architecture, prototypes and results to research conducted or products developed by others may not be as fruitful as when directly comparable existing work had been found. Our ideas and prototypes integrate features that can be found in several commercially available products or open-source software, but we think the novelty lies in the integration of these features.

The development of computers, the Internet and mobile and ubiquitous technologies has brought out many interesting topics of research during the last decades. Here, we have concentrated only on tiny fragments but have been faced with challenging questions nonetheless. Our topmost research issues have included application frameworks on mobile devices and dynamic protocol and connectivity management with the help of middleware. The Wellness application only acted as a special - although interesting - application scenario within this research.

6.1. PnPAP

The increasing importance of middleware has been noted by others, as well. For example, Raatikainen argues in [70] that the requirement of ever-faster service development and deployment implies the introduction of various application frameworks. Raatikainen continues by presenting middleware as a commonly proposed solution. We agree. As mobile and ubiquitous computing has developed and gained popularity rather quickly within the last decade, Software Development Kits (SDK) and development environments for many such devices have had difficulties keeping up with the pace. Resource constrained terminals often cannot execute programs developed with tools targeted for desktop environments, but they require their own set of tools, kits and APIs instead. Therefore, the potential of new hardware and software features is easily constrained by the lack of proper tools and documentation to utilize them in our own software development. The issue can be diminished by developing all the offered services to developers. Another possibility is to increased the abstraction level of programming by introducing new or updated APIs or even support for new, more efficient programming languages.

We have seen that the Series 60 platform, which we have the most personal experience of, has improved greatly in many of the aforementioned ways. However, this does not eliminate the need for middleware. It is the nature of SDKs to provide only the most basic building blocks that more advanced blocks can be developed on. There are so many different technologies, protocols and formats that SDKs will never be able to support them all. To enable efficient programming, existing advanced building blocks for features missing in the platform should be as readily available as possible. This is what application frameworks and middleware are all about.
Many middleware frameworks are provided or at least supported by major companies or semi-independent organizations financed by those companies or that at least have company representatives. Examples include the Common Object Request Broker Architecture (CORBA) by the Object Management Group (OMG) and the Distributed Component Object Model (DCOM) by Microsoft. Some also regard the Java Micro and Enterprise Editions as middleware by Sun Microsystems. Existing middleware platforms typically aim at providing a generic solution to building distributed systems, and developers can use the services provided by middleware to ease solving problems typically present in the process. However, the problem is that most of the platforms are incompatible, because they use a protocol or protocols developed for that specific middleware alone.

There are solutions to this problem. Some solutions include gateways in between the incompatible systems to enable interoperation \cite{26, 71}, whereas others propose that the all nodes should be able to ‘speak these different languages’ \cite{72}. PnPAP relies on the latter alternative. Mobile devices do have limited resources, but we claim that at least the modern ones are able to support the optimal use of multiprotocol and multiconnectivity, because it can in fact decrease the amount of used network bandwidth at the cost of added calculation. However, it studies have shown that calculation is many times cheaper in terms of battery consumption than using a wireless connectivity, and battery is perhaps even scarcer than processing power or memory in current mobile terminals.

One existing example of a multi-protocol application for mobile terminals is the Agile Messenger application \cite{73}. It is an Instant Messaging application with support for many different IM protocols, including MSN, AOL, Yahoo!, ICQ, Google Talk and XMPP. This resembles PnPAP in that compatibility is increased, but PnPAP provides expandability by being able to download a new protocol library and take it into use at run-time. On top of that, PnPAP can serve any number and type of applications, which makes it significantly more flexible. Shareaza \cite{74} is a P2P application with file sharing support for EDonkey2000, Gnutella, Gnutella (G2) and BitTorrent P2P networks. It has no dynamic protocol management and it is for desktop computers with the Windows operating system. It is open source, however. The program code could possibly be reused for a mobile implementation.

Desktop computers are often equipped with a fixed broadband Internet connection, and thereby require no intelligent connectivity management. Mobile IP based connectivity management solutions are commercially available for laptop computers. The multiple connectivities that have been available in mobile phones have been on the user’s manual control thus far. However, we have found that this is about to change. According to released information on Symbian OS v9.5 \cite{75}, the new release features IP bearer mobility which enables automatic switching between 3G and WLAN connections. This will ease the connectivity management task of PnPAP, but the use of Bluetooth or WLAN in adhoc mode should still be preferred instead of 3G or WLAN in architecture mode if the peers are at close range. In Symbian OS v9.5, this is most likely left for the user to control.

PnPAP probably never will be released for the public, but similar kind of functionality will probably be offered by operating systems of devices, standard programming APIs from device manufacturers or third-party middleware and other software frameworks. As time goes by, current state-of-the-art features become older and they become more widely supported even by lower-end devices.
and the complexity related to their use slowly shifts from users and/or application developers to lower levels such as operating systems or even hardware.

6.2. Wellness

The idea behind the Wellness application started off from personal interests. We wanted to be able to track workouts in a more modern way than just by using a pen and paper. The concept first included only exercises at the gym, but shortly the idea became more generic, enabling Wellness to be used for almost any sports to plan exercises and monitor progress. The whole application, including the GUI, was designed using personal background knowledge and experience of mobile applications and different sports.

For example, state-of-the-art heart rate meters include similar features as Wellness does. Besides showing the user’s heart rate, they can include programming support for different heart rate zones within a workout, and memory, the contents of which can be downloaded to a PC or a web application after a workout. Then, statistical information is updated and the workout plan can be updated accordingly if needed. For example, the Polar Personal Trainer web service seems to function like we described. In analyzing training effect and progress, desktop computers are obviously better than mobile terminals. A larger display, and a full-sized keyboard with a mouse can ease the task considerably. Wellness did not contain very effective tools to track progress. The users were only able to see how a specific exercise of their own or someone else proceeded. It was difficult to see if there was any progress between consecutive workouts, which is certainly something to improve on.

Wellness resembles the FRWD Mobile Player in that it is capable of showing workout related information. However, the Mobile Player includes only fixed heart rate and speed target values with no program support. Furthermore, the Mobile Player does not store or send any information, but relies on the sports computer instead to record the workout. The last generation (F-Series) sports computer itself resembles the service that was designed by Hyvönen in [43]. There was no way for the user to monitor the intensity, for example, during the training. This fact is shared by the two solutions. The current versions of the FRWD sports computer rely on a mobile phone to show the information in real-time (the B and older F series), or a wrist display similar to a heart rate meter (the W series). Visualization of information is done more elegantly in the Mobile Player used with the B and F series than in Wellness. The Replayer PC software, targeted for analysis after the training, even supports Google Earth to show the user’s route on the map of the globe. The Replayer is also capable of replaying a maximum of 5 records simultaneously, allowing the user to compare the performances. A separate, professional version of the Replayer includes more advanced analysis. The Genie system by Pirttikangas et al. [42] also includes analysis tools, but the system differs from Wellness by not allowing to guide the user during exercise and by lacking peer group features. Hyvönen only concentrates on the gathering of information with a Wireless Sensor Network (WSN), and analysis of the gathered information is left out of scope.

During our end-user testing, people found it difficult to edit workout programs using Wellness. In fact, at the start of the evaluation we mentioned about being
able to write programs with a text editor on any computer and then download the text files to the mobile phone via DC++. This would have also enabled backing up the information from the phone to a PC. We assume these options were hardly used, however. It would seem that an integrated planning and progress tracking service for use on desktops such as the Polar Personal Trainer is in fact an excellent solution and also contains a workout calendar to plan the training times beforehand. However, it may be more difficult to implement heart rate meter applications useful for workouts at the gym, because the user interface allows for even less output and input than a mobile phone does. Telling the user what exercise to take next and inputting repetitions, not to mention editing the plan, are probably not things in which heart rate meters would excel. Thus, mobile phones or even specialized equipment other than heart rate meters do have potential for use in training.

Including sports professionals and normal end-users to design the application from early on in the project would have probably been useful. The user evaluation results showed that the respondents found the user interface difficult. This may be partly due to the lack of instructions, but it may also be the result of an excessively generic nature of the user interface, which leads to too much information on the screen and too many options to choose from during workout. It is possible that a set of more focused applications on PnPAP - one for gym workouts, one for jogging, for example - could have provided better usability. After all, the best user interfaces are ones that do not require reading a manual before being able to use them.

Scarcity of comparable user evaluation data makes it difficult to contrast Wellness to another same kind of application from the user satisfaction point of view. However, we were disappointed with the results of the evaluation ourselves. The results indicate that the peer group related features were not very interesting, as they were rarely used. A major issue of our prototype was that it did not allow PnPAP to run without a connection to the FRWD sports computer over Bluetooth. This definitely should have been enabled. In any case, this restriction probably lowered the score for peer group related functionality, as there were no peers to test the features with in the peer group. Furthermore, the user should be able to control if PnPAP provides peers with access to shared resources even when no PnPAP application is running.

The instability of Wellness was well noted by all of our respondents. We did note possible problems before starting the end-user tests, but as with many software development projects, we had already overran our deadlines significantly. The volunteers had been eager to start testing for long, and preliminary tests for the software produced reasonable results. However, we had only tested the application ourselves before starting the evaluation which was a mistake. Instability and usability issues were noted very early after the evaluation began. As everybody was disappointed with the performance of the application, the results for all questions may have decreased. Based on the evaluated version, the end result was that Wellness seemed to fail in its goal of motivating people to exercise more. The users also thought the costs were greater than the benefit of using the application. However, due to the issues present in the prototypes and probable imperfections with asking the right questions and otherwise executing the evaluation, the results may not give a totally accurate view. Therefore, testing and fixing possible problems in time should have been preferred instead of completing
all features. Now, the results of the questionnaire have little or no value in telling if the idea of Wellness has potential. In fact, according to Nielsen [69, p. 209], user statements cannot always be considered reliable. Instead, the users’ true behavior should be investigated.

What is positive in the results of our questionnaire, however, is that the users did not feel concerned about sharing their context information in the peer group. Therefore, there may be potential in peer groups, but the features must be implemented differently. It may be that the mobile application should only be able to show, monitor, distribute and record the exercises, but planning, monitoring and analysis of information should be left to desktop computers. After all, not everything suits mobile use well. Some things are more convenient done differently. The mobile application could still include all the functionality to act as a complementary method.

Considering our implementation, Wellness was designed to utilize PnPAP from the start. It fit the scope of peer group centric file and context sharing perfectly. However, if PnPAP had not been available and no middleware serving many applications had not been needed, we probably would have implemented more features within the application itself. This would have saved us the trouble of implementing the client-server architecture, which hindered PnPAP prototype development many times. The rest of the classes probably would have still been needed, thus implementing Wellness without a separate middleware could be done with minor changes. In any case, having well-defined interfaces between different subcomponents of software raises the modularity of the software, which increases code reuse. Dividing the architecture in smaller parts also helps estimating required development resources and distributing the work in a project. Also, if dynamic linking is used for libraries, updating the executables can be done in a more fine-grained fashion. For example, when a bug is found and a new DLL is released that fixes the problem, only the updated DLL needs to be downloaded into the client devices. All other executables can potentially remain, which saves time and bandwidth in the update process.

### 6.3. Future Challenges

Many research questions related to our concepts were left unanswered. For example, we designed PnPAP to support dynamic control of protocols and connectivities. We did not, however, explain how the optimal ones are chosen in practice, and how the source of new libraries is determined. PnPAP intercommunication and state machines were brought up as possible solutions that have not been completed entirely, however.

The middleware required all PnPAP applications to implement specific functionalities. For example, a login screen needed to be present in every application. A better way could be to transfer these functionalities in PnPAP to further minimize required application development effort. PnPAP could have a settings application that would also enable the user to choose different protocol and connectivity management preferences, such as specifying the connectivities that PnPAP is allowed to use. Then, the user could restrain PnPAP from using connections that cost money.

PnPAP should optimally support communication in the absence of a commu-
communication architecture with ad-hoc networking via Bluetooth, for example. We did have the Bluetooth connectivity available but its function was confined to reading information from context data sensors. Another limitation of PnPAP was that joining multiple peer groups simultaneously was not supported. This could be an important feature in practice, but, for it to work, sharing files should be peer group specific. Our implementation shared the same files in any peer group that was joined.

Security and first-class reliability were advisedly left out of scope of our prototypes, as we originally only targeted limited proof-of-concept use. Implementing both could have affected the design, however, which should be taken into consideration during future work. Encrypted communication and support for anonymity should be supported on a more serious implementation. Stability and controlled processing of possible errors would have significantly increased the value of our user evaluation. Security features could include also anti-tampering protection of workout programs and results. The source of information should also be assured with proper methods.

Wellness should be developed further according to our more recent observations. As discussed earlier, the features for planning and sharing workout programs and analyzing results could be implemented as a desktop or a web application instead. The mobile phone would then mostly be used for the features that require mobility, instead of doing everything on the mobile terminal. Interfacing Wellness with gym equipment from Hur [76], for example, would make for an interesting future use case.
7. SUMMARY

In this Master’s Thesis, we presented two novel concepts: the Plug-and-Play Application Platform (PnPAP) and the Wellness application. PnPAP is a middleware architecture that provides easy-to-use communication services to applications with the help of a high-level API. Wellness, on the other hand, is a mobile application that features planning, monitoring, and recording physical workouts and sharing related information between peers.

To provide background information and to be able to justify choices made later on, we started off by introducing the state-of-the-art of related technology. Distributed computing and distributed system models were first presented, and the P2P model was discussed in more detail. The different generations of P2P were introduced including a comparison of the architectures. Middleware was presented as an important means to ease and speed up application development. Special characteristics of mobile computing were discussed next, with references to existing work related to mobile P2P. The first and third generation architectures were found especially suitable for mobile devices. As another topic of our state-of-the-art survey, we talked about wellness technology, focusing our presentation on devices and applications related to our Wellness application concept. Compared to our concepts, we found no similar products or existing research available. However, similar features were included in many devices, middleware and applications.

Then, we moved on to discussing our two concepts in more detail. PnPAP, developed as a joint effort in the Application SuperNetworking (All-IP) project at the University of Oulu, had already been implemented as a prototype for the Symbian/Series 60 platform and presented in many international publications, but in a relatively general way. We wanted to discuss PnPAP on a more detailed level. We took the Wellness application as a special application scenario and used common software development methods to define the use cases and requirements for the application. As we wanted Wellness to utilize the PnPAP middleware as well as to verify the services PnPAP offers against real-life application needs, we used the completed analysis of the Wellness application to divide the responsibilities of implementing the use cases between PnPAP and the application. This way, we produced the real-life use cases and preliminary requirements for PnPAP.

Next, the high-level PnPAP API was designed based on the use cases, and it was found that our existing PnPAP prototype already included the required services for the needs of the Wellness application. We moved on to present the detailed design for PnPAP and Wellness.

We performed many tests against the prototypes. We first tested them for compliance with the use cases and requirements. Most of the tests passed. We then started discussing the benefits gained in application development by utilizing PnPAP, and it was estimated that development resource savings between 20% and 60% may be reached. The feasibility of PnPAP dynamic protocol switch was evaluated by measuring the related additional delays in practice, and the conclusion was that the delay of downloading a new protocol library using GPRS was approximately 10 seconds. To eliminate this added delay during a user-requested service, it was proposed that basic protocol and connectivity libraries should be included in the PnPAP installation package. It was also suggested that new or updated libraries could be downloaded automatically when the terminal
was otherwise idle. The dynamic protocol switch was found useful in a separate test.

To further analyze the concepts and our prototypes, we arranged an end-user evaluation in collaboration with two other Master’s Thesis workers. The evaluation consisted of an opening meeting where the users performed a few tasks with the application and answered usability related questions on their questionnaire sheets. The second part of the evaluation lasted for one week and included free use of the application with loaned equipment consisting of an S60 mobile phone and a FRWD F-500 sports computer that was required by Wellness. During the week, the users were asked to use Wellness for their workouts. Afterwards, the respondents replied to the rest of the items on their questionnaire sheets and returned the sheets and the equipment.

Due to us having only three F-500 sports computers available, we organized the evaluation in groups of three people. The number of groups was 5, which means we had 15 respondents in total. The results were not encouraging, although the users did think the idea of Wellness was good. At least some of the negatively toned responses can be explained by the instability of the application, which was caused by lack of end-user testing before the evaluation. Wellness implementation deadlines had also been overrun significantly, and we needed to begin the tests regardless of bugs. In the end, we could not be sure whether the user evaluation results actually signalled anything more than flaws in the prototypes and execution of the evaluation. In any case, the users did consider the concepts good. Also, our other types of testing and measurements showed that our ideas are feasible. However, many issues were left as future work, which was discussed in the end.
8. REFERENCES


[57] Nokia (read 15 May 2007), S60 platform sdks for symbian os, for c++. URL: http://www.forum.nokia.com/info/sw.nokia.com/id/4a7149a5-95a5-4726-913a-3ef21eb65a5/S60-SDK-0616-3.0-mr.html.


[60] Unknown (read 18 May 2007), Open direct connect. URL: http://sourceforge.net/docman/?group_id=36589.


[74] Unknown (read 19 May 2007), Shareaza 2.2.5.0 beta. URL: http://www.shareaza.com/.


9. APPENDICES

1. Use Cases for the Wellness Application
2. Requirements for the Wellness Application
3. PnPAP API
4. PnPAP Design Diagrams
5. Wellness Design Diagrams
6. Shared Diagrams
7. User Evaluation Response Forms
8. Responses to User Evaluation Questionnaire
Use Case 1: Manage peer groups

**Actors:** User Alice

**Preconditions:** Alice has started Wellness Application and logged into PnPAP.

**Description:** Alice wants to create a new peer group and join it. She selects the “peer group” function from a menu and chooses to create a new Peer Group for herself and her friends. She defines the group name and password (Exceptions [1], [2] and [3]). After creating the peer group, she selects to join the peer group. She then invites her friends to join the group, e.g. by calling them.

**Exceptions:** [1] The device is out of memory: Alice is asked to close unnecessary applications and try again. [2] No connection to the network can be established. Alice is notified. [3] Peer Group with same name already exists: Alice is asked to input another name.

**Post-conditions:** Alice has created a new peer group and joined it. She can now start working in the peer group.

Use Case 2: Create new exercise programme

**Actors:** User Alice

**Preconditions:** Alice has started Wellness Application and logged into PnPAP.

**Description:** Alice wants to create a new exercise programme. She chooses “exercise programmes” and “create new” features from a menu. An empty exercise programme is created and opened to the modifying state (Exception [1]). Operation then continues as described in Use Case 2.

**Exceptions:** [1] The device is out of memory: Alice is asked to close unnecessary applications and try again.

**Post-conditions:** Alice has created an exercise programme and is ready to take the programme into use or share it.

Use Case 3: Modify exercise programme

**Actors:** User Alice

**Preconditions:** Alice has started Wellness Application and logged into PnPAP. She has at least one exercise programme stored on her mobile phone.

**Description:** Alice wants to modify an exercise programme. She either creates a new programme (described in Use Case 2) or selects to modify an existing one and selects the programme to be modified. The existing information is shown and Alice can insert new exercises and move or remove ones. Alice can specify the amount of repetitions or time periods for each training item.
possibly with heart rate limits. Alice then saves the modified programme, and she inputs a filename for the programme. The programme is then saved (Exceptions [1] and [2]). If a programme with the same name already exists, Alice is asked to confirm overwriting the existing exercise programme file.

Exceptions: [1] The device is out of storage space: Alice is asked to remove unnecessary files and try again. [2] The device is out of memory: Alice is asked to close unnecessary applications and try again.

Post-conditions: Alice has modified an exercise programme and is ready to share it or to start working out using the programme.

Use Case 4: Share exercise programme

Actors: User Alice

Preconditions: Alice has started Wellness Application and logged into PnPAP. She has at least one exercise programme stored on her mobile phone.

Description: Alice wants to share an exercise programme. She selects “exercise programmes” and a list of current exercise programmes is shown. Alice can then select the programme she wants to share. (Exception [1]). The same function can be used to disable share.

Exceptions: [1] The device is out of memory: Alice is asked to close unnecessary applications and try again.

Post-conditions: Alice has shared the exercise programme and others in the peer group can now find and download it.

Use Case 5: Search & download exercise programme

Actors: Users Alice, Bob

Preconditions: Alice and Bob have logged into PnPAP. Alice has started Wellness Application and Bob has shared at least one exercise programme in the peer group that Alice is in.

Description: Alice selects to search for exercise programmes in the current peer group. She selects “exercise programmes” and the search function and inputs keywords. Search is then performed and the search results are listed (Exception [1]). If no keywords are given, all programmes are listed. Alice finds a good programme from Bob’s device and selects to download it. The download begins (Exception [2]). Alice is notified when the download is completed. (Exceptions [3] and [4]).

Exceptions: [1] No connection to the peer group can be established. Alice is notified. [2] The device is out of storage space: Alice is asked to remove unnecessary files and try again. [3] The connection between Alice and Bob is lost before the download is completed: Alice is notified. [4] The device
is out of memory: Alice is asked to close unnecessary applications and try again.

**Post-conditions:** Alice has downloaded the exercise programme from Bob and can now share or modify it or use it in an exercise.

---

**Use Case 6: Work out**

**Actors:** User Alice

**Preconditions:** Alice has started Wellness Application and logged into PnPAP. Alice has at least one suitable exercise programme stored on her mobile phone.

**Description:** Alice wants to work out. She selects “exercise programmes” and selects a suitable one. She then selects “work out” and the application switches into exercise mode. The application starts to share Alice’s exercise data including location, heart rate etc. to the current peer group (Exception [1]). The application also starts to write the exercise data locally into persistent storage to make it accessible later (Exception [2]). The application acts according to the exercise programme that Alice selected and shows Alice the next exercises. If exercise times (e.g. in interval training) are specified in the programme, the application notifies Alice whenever a new milestone has been reached. The application also shows the current heart rate with limits and alerts Alice if it is out of bounds if Alice is using the FRWD sports computer with a heart rate transmitter belt. Alice can work out effectively and keep track of her results easily by inputting possible repetition counts into the application. (Exception [3]).

**Exceptions:** [1] No connection to the peer group can be established. Alice is notified. [2] The device is out of storage space: Alice is asked to remove unnecessary files and try again. [3] The device is out of memory: Alice is asked to close unnecessary applications and try again.

**Post-conditions:** Alice has had fun in an efficient workout and she can now access the results by viewing her history data with the application.

---

**Use Case 7: See own exercise history**

**Actors:** User Alice

**Preconditions:** Alice has started Wellness Application and logged into PnPAP. Alice has worked out at least once using the application.

**Description:** Alice wants to see her training results from past exercises. She selects “exercise history” from the menu and finds a list of the work outs stored into her mobile device. She selects an exercise done a week ago. She can then see the specific exercises she made with repetition counts, time periods and heart rates. (Exception [1]).
Exceptions: [1] The device is out of memory: Alice is asked to close unnecessary applications and try again.

Post-conditions: Alice has found the results of her last workout and she can now aim higher in her next training.

Use Case 8: Share own exercise history

Actors: User Alice

Preconditions: Alice has started Wellness Application and logged into PnPAP. Alice has worked out at least once using the application.

Description: Alice wants to share her training results from past exercises. She selects “exercise history” from the menu and finds a list of the work outs stored into her mobile device. She selects an exercise done a week ago. She then selects “share”. She scrolls down to find another exercise that is already shared. As it is an old one, she disables the share of that exercise and removes the recording. (Exception [1]).

Exceptions: [1] The device is out of memory: Alice is asked to close unnecessary applications and try again.

Post-conditions: Alice has shared the results of her last workout and her friends can see the results she got during that exercise.

Use Case 9: Follow others’ exercise in real time

Actors: Users Alice, Bob

Preconditions: Alice and Bob have started Wellness Application and logged into PnPAP. Alice and Bob have joined the same peer group. Bob is currently doing a workout using the application.

Description: Alice wants to see what is going on in her peer group. She selects to see a list of the peers currently in the peer group (Exception [1]). She sees that Bob is doing a workout. She wants to find out more accurately the context Bob is in, thus she selects Bob and sees the specific exercise Bob is doing with his context data including heart rate (Exceptions [2] and [3]).

Exceptions: [1] No connection to the peer group can be established. Alice is notified. [2] No connection to Bob’s device can be established. Alice is notified. [3] The device is out of memory: Alice is asked to close unnecessary applications and try again.

Post-conditions: Alice has found out that Bob is still in an early phase of his workout. She decides to go and join in.
Use Case 10: See others’ exercise history

**Actors:** Users Alice, Bob

**Preconditions:** Alice and Bob have started Wellness Application and logged into PnPAP. Alice and Bob have joined the same peer group. Bob has done at least one workout using the application and shared the results.

**Description:** Alice wants to see how people in her peer group have developed. She selects to see a list of the peers currently in the peer group (Exception [1]). She sees Bob in the list. As Bob has gained good results lately, she wants to see if he has still kept up the same pace. She selects Bob and sees a list of Bob’s shared exercises (Exception [2]). She then selects an exercise and then can browse through all the details. (Exception [3]).

**Exceptions:** [1] No connection to the peer group can be established. Alice is notified. [2] No connection to Bob’s device can be established. Alice is notified. [3] The device is out of memory: Alice is asked to close unnecessary applications and try again.

**Post-conditions:** Alice has found out that Bob is in fact still making good results by switching the exercise programme. She decides to contact Bob to get guidance in efficient workouts.

Use Case 11: Start interaction from another application

**Actors:** Users Alice, Bob

**Preconditions:** Alice is using some PnPAP based application. Bob has started Wellness Application and logged into PnPAP. Alice and Bob have joined the same peer group. Bob is currently working out and has allowed Alice to access his context data.

**Description:** Alice is using a chat application to try and start discussing with Bob. However, the application shows Bob’s presence information and it seems he is currently unavailable for chatting. Alice would like to see what he is doing more specifically. She opens a menu in the chat application to see a list of interactions Bob and Alice are allowed to have (Exception [1]). Alice is shown with an option of following Bob’s workout and she selects it. The Wellness application pops up (Exception [2]) and immediately starts showing real-time context data from Bob’s workout (Exception [3]).

**Exceptions:** [1] No connection to the peer group can be established. Alice is notified. [2] The device is out of memory: Alice is asked to close unnecessary applications and try again. [3] No connection to Bob’s device can be established. Alice is notified.

**Post-conditions:** She can see that Bob’s workout is only in the beginning. She decides to go and join him.
Use Case 12: Access notice board (optional)

**Actors:** Users Alice, Bob

**Preconditions:** Alice and Bob have started Wellness Application and logged into PnPAP. Alice and Bob have joined the same peer group.

**Description:** Bob has just shared a new revolutionary exercise programme and wants others to find it. He selects “notice board” and “write message”. He then writes a nice message about the new programme and sends it to the notice board (Exception [1]). Alice selects “notice board” and “read messages”. She finds Bob’s message (Exception [2]) and goes on immediately to download Bob’s new programme out. (Exception [3]).

**Exceptions:** [1] No connection to the peer group can be established. Bob is notified. [2] No connection to Bob’s device can be established. Alice is notified. [3] The device is out of memory: Alice is asked to close unnecessary applications and try again.

**Post-conditions:** Bob has notified others with a persistent notice board message and Alice has read it and found it useful.
# Functional Requirements (FR)

<table>
<thead>
<tr>
<th>#</th>
<th>Requirement</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The application supports operating in peer groups</td>
<td>MUST</td>
</tr>
<tr>
<td>2</td>
<td>The application supports joining open and closed peer groups</td>
<td>MUST</td>
</tr>
<tr>
<td>3</td>
<td>The application supports planning workouts for many types of sports</td>
<td>MUST</td>
</tr>
<tr>
<td>4</td>
<td>The application supports creating, editing and removing work out programs</td>
<td>MUST</td>
</tr>
<tr>
<td>5</td>
<td>A terminal can store many work out programs</td>
<td>MUST</td>
</tr>
<tr>
<td>6</td>
<td>A work out program can have many sets of exercises</td>
<td>MUST</td>
</tr>
<tr>
<td>7</td>
<td>An exercise can be given a target in repetitions, length (m) or time (s)</td>
<td>MUST</td>
</tr>
<tr>
<td>8</td>
<td>An exercise can be given heart rate and speed limits</td>
<td>MUST</td>
</tr>
<tr>
<td>9</td>
<td>Work out programs can be shared between peers</td>
<td>MUST</td>
</tr>
<tr>
<td>10</td>
<td>Work out programs can be searched and downloaded from other peers</td>
<td>MUST</td>
</tr>
<tr>
<td>11</td>
<td>Searches can include keyword filtering</td>
<td>MUST</td>
</tr>
<tr>
<td>12</td>
<td>The user can start a work out using the work out programs on her terminal</td>
<td>MUST</td>
</tr>
<tr>
<td>13</td>
<td>The user can start a work out without a work out program (i.e. quick start)</td>
<td>MUST</td>
</tr>
<tr>
<td>14</td>
<td>The user can add new exercises to the program during the work out</td>
<td>MUST</td>
</tr>
<tr>
<td>15</td>
<td>During workout, the application keeps track of the result (time or length) automatically</td>
<td>MUST</td>
</tr>
<tr>
<td>16</td>
<td>When entering an exercise result, the user can edit it (reps, time, length) manually</td>
<td>MUST</td>
</tr>
<tr>
<td>17</td>
<td>During workout, the application can show done and planned exercises with targets and results</td>
<td>MUST</td>
</tr>
<tr>
<td>18</td>
<td>The application shows the target and current result within a workout</td>
<td>MUST</td>
</tr>
<tr>
<td>19</td>
<td>The showed results include heart rate and speed with limits, altitude, temperature and statistics</td>
<td>MUST</td>
</tr>
<tr>
<td>20</td>
<td>The user is informed of heart rate and speed limit &quot;violations&quot; using the screen and the speaker</td>
<td>MUST</td>
</tr>
<tr>
<td>21</td>
<td>The user is informed of reaching the target length or time using the screen and the speaker</td>
<td>MUST</td>
</tr>
<tr>
<td>22</td>
<td>The application can show a list of peers currently working out</td>
<td>MUST</td>
</tr>
<tr>
<td>23</td>
<td>The user’s workout can be followed during the training by a peer in the user’s PG using the application</td>
<td>MUST</td>
</tr>
<tr>
<td>24</td>
<td>Work out data is stored during the workout</td>
<td>MUST</td>
</tr>
<tr>
<td>25</td>
<td>Work out data can be shared between peers</td>
<td>MUST</td>
</tr>
<tr>
<td>26</td>
<td>The application features a list of the user’s past work outs</td>
<td>MUST</td>
</tr>
<tr>
<td>27</td>
<td>The application can replay a past work out of the user</td>
<td>MUST</td>
</tr>
<tr>
<td>28</td>
<td>The application can show a list of peers sharing past workout data</td>
<td>MUST</td>
</tr>
<tr>
<td>29</td>
<td>The application can show the shared work out history of a selected peer</td>
<td>MUST</td>
</tr>
<tr>
<td>30</td>
<td>The application can replay a shared past work out of a peer</td>
<td>MUST</td>
</tr>
<tr>
<td>31</td>
<td>The data shown (e.g. types of workout information) is independent of the source of the data</td>
<td>MUST</td>
</tr>
<tr>
<td>32</td>
<td>The application does not lose stored work out data on crash</td>
<td>SHOULD</td>
</tr>
<tr>
<td>33</td>
<td>The application enables continuing work out after crash</td>
<td>SHOULD</td>
</tr>
<tr>
<td>34</td>
<td>The application supports creating open and closed peer groups</td>
<td>SHOULD</td>
</tr>
<tr>
<td>35</td>
<td>The application supports activating interactions by other applications</td>
<td>SHOULD</td>
</tr>
<tr>
<td>36</td>
<td>The application features a PG specific notice board with read and write operations</td>
<td>MAY</td>
</tr>
</tbody>
</table>
## Environmental Requirements (ER)

<table>
<thead>
<tr>
<th>#</th>
<th>Requirement</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The application works on a standard S60 2nd Ed. FP2 mobile phone</td>
<td>MUST</td>
</tr>
<tr>
<td>2</td>
<td>The application can utilize GPRS for communication between peers</td>
<td>MUST</td>
</tr>
<tr>
<td>3</td>
<td>The application supports reading sensor data from FRWD F-500 via BT</td>
<td>MUST</td>
</tr>
<tr>
<td>4</td>
<td>Compatible S60 phone and FRWD F-500 are the only devices required</td>
<td>MUST</td>
</tr>
<tr>
<td>5</td>
<td>The application uses open, standard communication protocols</td>
<td>SHOULD</td>
</tr>
<tr>
<td>6</td>
<td>Content sharing is compatible with at least one existing PC application</td>
<td>SHOULD</td>
</tr>
<tr>
<td>7</td>
<td>The application can utilize BT and WLAN for communication between peers</td>
<td>MAY</td>
</tr>
</tbody>
</table>

## Usability Requirements (UR)

<table>
<thead>
<tr>
<th>#</th>
<th>Requirement</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The application is easy to use</td>
<td>MUST</td>
</tr>
<tr>
<td>2</td>
<td>The application can be used with minimal training</td>
<td>MUST</td>
</tr>
<tr>
<td>3</td>
<td>The application responds to user input in less than 1 s</td>
<td>MUST</td>
</tr>
<tr>
<td>4</td>
<td>Operations longer than 1 s show a wait dialog/progress bar</td>
<td>MUST</td>
</tr>
<tr>
<td>5</td>
<td>The amount of required manual configuration is minimal</td>
<td>MUST</td>
</tr>
<tr>
<td>6</td>
<td>The work out program files are in human readable format</td>
<td>MUST</td>
</tr>
<tr>
<td>7</td>
<td>Searching completes within 30 s</td>
<td>SHOULD</td>
</tr>
<tr>
<td>8</td>
<td>Downloading (programs or results) typically completes within 1 min</td>
<td>SHOULD</td>
</tr>
<tr>
<td>9</td>
<td>Following a peer’s workout happens in near real-time (delay max dozens of s)</td>
<td>SHOULD</td>
</tr>
</tbody>
</table>
MPnPAPAppInterface

This is the class for use by applications to request services.

Generic

- TInt LoginL(const TDesC& aUsername, const TDesC& aPassword); /* Sync - Login to PnPAP. The idea is that the user enters this information into the application that then calls this method. This is how the user is authenticated for further resource access. */
- TBool AmILoggedIn(HBufC*& aUserName); /* Sync - Return if this or some other application has already logged into PnPAP. */

Peer group management

- TInt CreatePeerGroupL(const TDesC& aPgName, const TDesC& aPgType, const TDesC& aPgPassword); /* Async - Create a new peer group. The type can be, e.g. public or private. If private, a password is needed for authentication. */
- TInt JoinPeerGroupL(const TDesC& aMyName, const TDesC& aPgName, const TDesC& aPgPassword); /* Async - Join a peer group. A peer name is given to enable the user to have different names in different groups. If private, a password is needed for authentication. */
- TInt LeavePeerGroupL(const TDesC& aPgName); /* Async - Leave a peer group. */
- TInt GetPeerGroupMembersL(CDesCArray* aList); /* Sync - Get a list of peers in the current peer group(s). */
- TBool GetJoinedPeerGroupsL(CDesCArrayFlat*& aPeerGroupNames, CDesCArrayFlat*& aJoinNames); /* Sync - Get a list of peer groups that the user has joined, and a list of the user’s names in those groups. */

Interaction

- TInt SendMessageL(const CDesCArray& aPeers, const TDesC& aText); /* Async - Send a textual message to peers in the current peer group(s). If aPeers is empty, the message is sent to all peers in the current peer group(s). */
- TInt SetPermissionsL(const TDesC& aPath, TUint aPermission); /* Sync - Sets a file shared in the current peer group(s) or removes share. */
- TInt SearchContentsL(const TDesC& aTargetPeer, const CDesCArray& aSearchParameters); /* Async - Search for shared resources in the current peer group(s). If aTargetPeer is given, the search is targeted to that peer group. */
only. If aSearchParameters is empty, all the shared resources are returned.

- TInt DownloadToLocalL(const TDesC& aSrcPeer, const TDesC& aContentHash, const TDesC& aLocalPath); /* Async - Download a file. The aContentHash can be a file name, but also a UID if the protocol supports them. */

- TInt GetSupportedInteractionsL(const TDesC& aPeer, CDescCArray* aList); /* Sync - Return a list of supported interactions with the given peer. aList contains interaction descriptions that can be shown to the user and input as parameter to the StartInteractionL function. */

- TInt StartInteractionL(const TDesC& aRemoteUserName, const TDesC& aParameters); /* Async - Start an interaction with another peer. The currently available interactions given in aParameters are listed in the peer’s context information. */

**Context related**

- TInt GetContextDataL(TBool aContinuous); /* Async - Get context data of peers in the current peer group(s) that have allowed the user to. If aContinuous is false, only one asynchronous return is given. */

- TInt StopGettingContextDataL(); /* Sync - Stop getting context data. */

- CEntityList* GetContextDataLC(); /* Sync - Get cached context data of peers in the current peer group(s) that have allowed the user to. */

- TInt RequestVisibilityChangeL(const CEntityList& aRemoteUsers); /* Async - Request other peers in current peer group(s) to allow the user to examine their context information. The pieces of information whose access rights are requested are listed in the context data of aRemoteUsers. */

- CEntityList* GetMyVisibilityLC(); /* Sync - Get the local user’s visibility settings. */

- TInt SetMyVisibilityL(const CEntityList& aRemoteUsers); /* Sync - Set the local user’s visibility settings. */

- TInt RequestVisibilityChangeReplyL(TBool aOk); /* Sync - Respond to a request from a peer to alter the local user’s visibility settings. */

- CContextData* GetMyContextDataLC(); /* Sync - Get the local user’s context data. */

- void UpdateMyContextDataL(const CContextData& aData); /* Sync - Update the local user’s context data. Typically, it is gathered by PnPAP automatically. However, this provides an override, e.g. the user’s presence can be set by an application. */
MPnPAPCallbackInterface

Next, this is the callback interface class for asynchronous functions.

- void RegisterReturn(TInt aFuncId, TAny* dataPtr); /* Return results from a preceding asynchronous function call to the PnPAP API, e.g. SearchContentsL. Each function has its own id, and the application receiving the results can then cast the dataPtr to a known object pointer. Transfers ownership of dataPtr to the application. */

- void RegisterErrorReturn(TInt aFuncId, const TDesC& aErrorMsg); /* Notify the application of an error that occurred in a preceding asynchronous function call to the PnPAP API. */

- void UnsetFlagAndDealloc(TInt aFuncId); /* Tell the application to free the memory that is reserved by results provided to the application by a RegisterReturn call. Typically, the application calls this itself. */

- TBool FlagValue(TInt aFuncId); /* Ask the application if its result buffers are empty, i.e. if it has handled the possible preceding results. */

- TAny* DataPtrValue(TInt aFuncId); /* Return the data pointer to the current results of a function. Mostly called by the application by itself. */

- void VisibilityUpdateL(CUser* aUser); /* Inform the application that a certain peer’s visibility has been changed to allow the local user to access her context data. Details of allowed context types are included in the aUser object. */

- void RequestVisibilityChangeL(CUser* aUser); /* Inform the application that a peer is requesting the local user to change his visibility settings to allow the peer to access her context data. Details of requested context types are included in the aUser object. */

- void ContextDataChangedL(CEntityList* aEntityList); /* Provide the application with context data from peers in the current peer group(s). Result from a preceding GetContextDataL call. */

- void ActivateInteraction(const TDesC& aRemoteUserName, const TDesC& aParameters); /* Request the application to activate an interaction with another peer. Typically, this should change the view of the application, e.g. show a chat view if a chat interaction is started. */
Figure A4.14. The class diagram of PnPAP Client/Server architecture.
Figure A4.15. The PnPAP server side class diagram.
Figure A4.16. The class diagram for the GPRS connectivity.
Figure A4.17. The class diagram for DC++ protocol.
Figure A5.18. The application architecture behind the Wellness Application.
Figure A5.19. State chart of the Wellness Application GUI.
Figure A6.20. The deployment diagram of Wellness and PnPAP.
Figure A6.21. The sequence diagram for Use Case 1.

Figure A6.22. The sequence diagram for Use Cases 4 and 8.
In case mode is passive, the results are sent through the hub.

The user wants to download a file.

In case mode is passive, RevConnectToMe is used.

From client connection when transfer complete.

CClientConnection handles the handshaking and the transfer.

Timer: delete client connection (deletes its connectivity).

GetPeerGroupMembersL

Only in Use Case 10

SearchContentsL

SearchContentsL

ListenUdpL

WriteL

UdpMessageReceived

EndListenUdp

Create new CConnectivityGprs and CClientConnection

Connect via TCP

DownloadToLocalL

DownloadToLocalL

$Search

$Search

$ConnectToMe

$ConnectToMe

$SR

$SR

Figure A6.23. The sequence diagram for Use Cases 5 and 10.
The user chooses a workout program. AppUI calls StartWorkoutL that deserializes the workout program.

PnPAP server deserializes the context data and stores it.

And so on...

Figure A6.24. The sequence diagram for Use Case 6.
The user enters the peer list view

Returns a list of peers in the DC++ hub

Ask for permission to exchange context (possibly from the user via RequestVisibilityChangeL)

Ask for continuous context data stream, user not involved

Context transferring continues...

Figure A6.25. The sequence diagram for Use Case 9.
Wellness is switched to foreground, opens the training view and starts showing the peer’s context.

And so on...

Figure A6.26. The sequence diagram for Use Case 11.
Appendix 7 User Evaluation Response Forms

OSA 1: Taustatiedot

1. Sukupuoli: Nainen □........Mies □

2. Ikä: 15-20 □........21-25 □........26-30 □........30-40 □........41+ □

3. Ammatti: ________________________________________________________________

4. Koulutus: ________________________________________________________________

5. Oletko aikaisemmin käyttänyt ns. älypuhelinta? En koskaan Usein
   □ 2□ 3□ 4□

6. Kuinka usein olet käyttänyt seuraavia matkapuhelinpalveluita?
   Internetin selaus älypuhelimella
   Käytön syy? En koskaan Usein
   □ 2□ 3□ 4□
   Työ □ huvi □ koulutus □ muu □

   GPRS/EDGE/3G
   Käytön syy? En koskaan Usein
   □ 2□ 3□ 4□
   Työ □ huvi □ koulutus □ muu □

   Pelien tai sovellusten lataus älypuhelimeen
   Käytön syy? En koskaan Usein
   □ 2□ 3□ 4□
   Työ □ huvi □ koulutus □ muu □

7. Montako älypuhelinsovellusta olet itse ostanut? __________________________________

8. Oletko aiemmin käyttänyt vertaisverkkosovelluksia (esim. Napster, DC++, Kazaa, Gnutella),
   PC:llä tai matkapuhelimella?
   Käytön syy?
   Millaisia sovelluksia?

   En koskaan Usein
   □ 2□ 3□ 4□
   Työ □ huvi □ koulutus □ muu □

Sovelluksiin liittyvät esitetokysymykset

9. Oletko aiemmin käyttänyt navigointisovelluksia?
   Käytön syy?
   Millaisilla laitteilla (matkapuhelin, erillislaitte, muu)?

   En koskaan Usein
   □ 2□ 3□ 4□
   Työ □ huvi □ koulutus □ muu □

10. Kuinka usein keskimäärin harrastat urheilua (poislukien hyötyliikunta kuten töihin pyöräily)?
    a. □ n. 2 kertaa kuukaudessa tai vähemmän
    b. □ n. 3 kertaa kuukaudessa
    c. □ n. kerran viikossa
    d. □ n. 2 kertaa viikossa
    e. □ n. 3 kertaa viikossa tai enemmän

11. Mitä urheilulajeja harrastat?

   ________________________________________________________________
   ________________________________________________________________
   ________________________________________________________________
12. Miksi urheilet (tavoitteet)? / Miksi et urheile?
______________________________________________________________________________

13. Millaisia apuvälineitä olet käyttänyt urheillessasi tai kuntoilun tulosten seurannassa (esim. sykemittari (merkki ja malli), kynä ja vihko)?
______________________________________________________________________________
______________________________________________________________________________

14. Mitä muita harrastuksia sinulla on?
______________________________________________________________________________
______________________________________________________________________________
______________________________________________________________________________
**WELLNESS: TEHTÄVÄT**

Tee tehtävät. Ympyröi mielipiteesi väitteistä ja kirjoita havaintosi ja kokemuksesi Wellness-sovelluksen käytöstä.

**Tehtävä 1: Sovelluksen käynnistäminen ja vertaisryhmään (DC++-hubille) liittyminen.**
- Käynnistä Wellness.
- Kirjoita nimesi tekstin "sip:" jälkeen, esimerkiksi "sip:Seppo@212.50.147.120".
- Kirjoita salasanaksi mitä tahansa, esimerkiksi "a".
- Paina OK. Sinut kirjataan sisään ja pääset päävalikkoon.
- Valitse päälukosta *Manage peer groups.*
- Valitse *Options* ja *Join.* Hubi lähettää kaksi tervetuloviestiä.

(Jos liittyminen ei onnistu, voit "puhdistaa" DC++-hubin liittymällä sille uudestaan eri nimellä: poistu sovelluksesta (myös NaviP2P:stä) ja kirjaudu uudelleen eri nimellä.)

**Väitteet:**

<table>
<thead>
<tr>
<th>Erä mieltä</th>
<th>Samaa mieltä</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Kirjautuminen oli helppoa.</td>
<td>1</td>
</tr>
<tr>
<td>2. Kirjautuminen oli nopea</td>
<td>1</td>
</tr>
</tbody>
</table>

**Tehtävä 2: Treeniohjelman hakeminen**
- Valitse päälukosta *Workout Programs.*
- Valitse *Options* ja *Search PG* (peer group).
- Syötä hakusanat, esim. “kuntosali” ja paina OK.
- Hakutuloksen ilmestyttyä valitse *Options* ja *Download.*

**Väitteet:**

<table>
<thead>
<tr>
<th>Erä mieltä</th>
<th>Samaa mieltä</th>
</tr>
</thead>
<tbody>
<tr>
<td>3. Hakeminen oli helppoa.</td>
<td>1</td>
</tr>
<tr>
<td>4. Hakeminen oli nopea</td>
<td>1</td>
</tr>
</tbody>
</table>

Havaintoja ja kokemuksia:
__________________________________________________________________
__________________________________________________________________

**Tehtävä 3: Harjoituksen aloittaminen.**
- Valitse päälukosta *Workout Programs.*
- Valitse *Options* ja *Start workout.*
- Aloittaaksesi harjoituksen osasuorituksen valitse *Options* ja *Start exercise.*
- Voit liikkua eri tietosivujen (oikea- vasen) ja osaharjoitusten (alas- ylös) välillä.

**Väitteet:**

<table>
<thead>
<tr>
<th>Erä mieltä</th>
<th>Samaa mieltä</th>
</tr>
</thead>
<tbody>
<tr>
<td>5. Harjoituksen aloittaminen oli helppoa.</td>
<td>1</td>
</tr>
</tbody>
</table>

Havaintoja ja kokemuksia:
__________________________________________________________________

**Tehtävä 4: Toisen käyttäjän harjoituksen seuraaminen.**
- Valitse päälukosta *Workout Results.*
- Selaa sellaisen käyttäjän kohdalle, jonka nimi alkaa ”sip:” (ei itsesi). Parhaillaan harjoittelevien käyttäjien nimen alla lukee ”Working out”.
- Valitse *Options* ja *Follow training.*

**Väitteet:**

<table>
<thead>
<tr>
<th>Erä mieltä</th>
<th>Samaa mieltä</th>
</tr>
</thead>
<tbody>
<tr>
<td>6. Harjoituksen seuraamisen aloitus oli helppoa.</td>
<td>1</td>
</tr>
<tr>
<td>7. Harjoituksen seuraamisen aloitus oli nopea.</td>
<td>1</td>
</tr>
</tbody>
</table>

Havaintoja ja kokemuksia:
__________________________________________________________________
WELLNESS: KYSYMYKSET KÄYTÖN JÄLKEEN

1. Arvioi miten monta tuntia _______ ja käyttökertaa _______ käytit palvelua yhteensä.
2. Miten monta kertaa käytit seuraavia toimintoja?
   Treeniohjelman luominen tai muokkaus _______ Treeniohjelman lataaminen muilta _______
   Palvelun käyttö harjoituksessa _______ Oman harjoitushistorian tutkiminen _______
   Toisen käyttäjän harjoituksen seuraaminen reaaliajassa _______ Toisen käyttäjän harjoitushistorian
tutkiminen _______

3. Arvioi mihin lajeihin ja mille tasolle palvelu soveltuu hyvin.
   ____________________________________________
   ____________________________________________
   ____________________________________________
   ____________________________________________

FRWD-laite maksaa 139 e ja sovelluksen vaatima S60-puhelin alkaen noin 200 e. Palvelun
käyttöä varten tarvitaan lisäksi datayhteys, esim. 10 e/kk (rajaton käyttö, Saunalahti).

Vääitteet:

4. Palvelun kokonaiskustannukset ovat kohtuulliset
   hyötyihin nähden. 1 □ 2 □ 3 □ 4 □
5. Urheilin keskimääräistä enemmän testijakson aikana. 1 □ 2 □ 3 □ 4 □
6. Jos saisin jatkaa palvelun käytöä, urheilisin enemmän
   myös jatkossa. 1 □ 2 □ 3 □ 4 □
7. Palvelu oli parempi kuin aiemmin käyttämäni
   tulosseurantametodi. 1 □ 2 □ 3 □ 4 □
8. Käyttämäni päätelaite sopii hyvin palvelun käyttöön. 1 □ 2 □ 3 □ 4 □
9. Palvelun käyttö oli helppoa. 1 □ 2 □ 3 □ 4 □
10. Olen tyytyväinen palveluun. 1 □ 2 □ 3 □ 4 □
11. Tietojeni jakaminen herätti minussa huolta. 1 □ 2 □ 3 □ 4 □
12. Jatkaisin mielelläni palvelun käyttöä. 1 □ 2 □ 3 □ 4 □

13. Mikä palvelussa oli parasta?
   ____________________________________________
   ____________________________________________
   ____________________________________________
   ____________________________________________

14. Mikä palvelussa oli huonointa?
   ____________________________________________
   ____________________________________________
   ____________________________________________
   ____________________________________________

15. Kehittämisideoita ja muita kommentteja?
   ____________________________________________
   ____________________________________________
   ____________________________________________
   ____________________________________________

16. Minkä kokonaisarvosanan antaisit palvelulle (arvosana 1-5)?
   ____________________________________________
## Appendix 8 Responses to User Evaluation Questionnaire

<table>
<thead>
<tr>
<th>Background information</th>
<th>Respondent (group.id)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.1</td>
</tr>
<tr>
<td>Sex</td>
<td>Female</td>
</tr>
<tr>
<td>Used a smartphone before</td>
<td>4</td>
</tr>
<tr>
<td>Used P2P applications before</td>
<td>1</td>
</tr>
<tr>
<td>Mobile Internet</td>
<td>4</td>
</tr>
<tr>
<td>Frequency of installing smartphone apps</td>
<td>1</td>
</tr>
<tr>
<td>Number of smartphone apps bought</td>
<td>0</td>
</tr>
<tr>
<td>Used heartrate meter before</td>
<td>Yes</td>
</tr>
<tr>
<td>Login was easy</td>
<td>4</td>
</tr>
<tr>
<td>Login was fast</td>
<td>4</td>
</tr>
<tr>
<td>Searching was easy</td>
<td>4</td>
</tr>
<tr>
<td>Searching was fast</td>
<td>2</td>
</tr>
<tr>
<td>Starting a workout was easy</td>
<td>4</td>
</tr>
<tr>
<td>Starting to follow workout was easy</td>
<td>3</td>
</tr>
<tr>
<td>Starting to follow workout was fast</td>
<td>4</td>
</tr>
<tr>
<td>After short exercises</td>
<td>225</td>
</tr>
<tr>
<td>Total instances of use</td>
<td>2</td>
</tr>
<tr>
<td>Creating/editing program (times)</td>
<td>2</td>
</tr>
<tr>
<td>Downloading program (times)</td>
<td>1</td>
</tr>
<tr>
<td>Working out using the service (times)</td>
<td>2</td>
</tr>
<tr>
<td>Examining own workout history (times)</td>
<td>1</td>
</tr>
<tr>
<td>Following peer's workout (times)</td>
<td>0</td>
</tr>
<tr>
<td>Examining peer's workout history (times)</td>
<td>1</td>
</tr>
<tr>
<td>Opinions</td>
<td>211</td>
</tr>
<tr>
<td>Total costs vs. benefit reasonable</td>
<td>2</td>
</tr>
<tr>
<td>Worked out more than usual</td>
<td>2</td>
</tr>
<tr>
<td>Would continue to work out more</td>
<td>1</td>
</tr>
<tr>
<td>Service was better than previous method</td>
<td>1</td>
</tr>
<tr>
<td>Device was suitable for service</td>
<td>3</td>
</tr>
<tr>
<td>Using the service was easy</td>
<td>3</td>
</tr>
<tr>
<td>I am satisfied with the service</td>
<td>2</td>
</tr>
<tr>
<td>Sharing my data aroused concern</td>
<td>2</td>
</tr>
<tr>
<td>I would like to continue using the service</td>
<td>1</td>
</tr>
<tr>
<td>Grade 1-5</td>
<td>3</td>
</tr>
</tbody>
</table>