W3S

WiFi 3G Share

Aalborg University
Electronic Computer Systems
Fall 2009 - Group 09gr750

Aalborg University
Institute of Electronic Systems

TITLE
WiFi 3G Share

PROJECT
Professional bachelor in
Computer Engineering
with speciality in
Communication Systems
final project

PERIOD
2nd Nov. 2009 - 7th Jan. 2010

GROUP 09gr750

Thomas Skinner-Larsen

Henning Gørtz Hørstrup

SUPERVISOR
Rasmus Olsen

NUMBER OF REPORTS
4

NUMBER OF PAGES
89

TOTAL NUMBER OF PAGES
104

Synopsis

The purpose of this project is to design and develop a platform for discovering and distributing services between mobile phones in an ad hoc network. The effectiveness of the platform is depending of the number of active participants. On the other hand, a high number of active participants would result possibility of additional discoveries made by the participants which increases the total energy consumed on the network. For such a platform a prototype is build. The prototype is based on preliminary demands, deployment and activity diagrams which are analyzied and designed on basis of UML 2.0. The working environments are Symbian S60 on the Nokia N95 8GB phones. The programming language used for developing the platform is Python for s60.

The functionality of the prototype platform is verified in the test chapter, and further more an analysis of the platform regarding how much energy the implemented software for the mobile phones is consuming with reactive or proactive communication.

The prototype platform is working according to what is analysied and designed, it is possible to discover and distribute services between mobile phones.

Aalborg Universitet
Institut for Elektroniske Systemer

TITEL
WiFi 3G Share

PROJEKT
Professionsbachelor
afgangsprojekt i
Computer Ingeniør
med speciale i
Kommunikationssystemer

PERIODE
2. Nov. 2009 - 7. Jan. 2010

GRUPPE 09gr750

_____
Thomas Skinner-Larsen

_____
Henning Gørtz Hørstrup

VEJLEDER
Rasmus Olsen

ANTAL RAPPORTER
4

ANTAL SIDER
89

TOTAL ANTAL SIDER
104

Synopsis

Formålet med dette projekt er at designe og udvikle en platform som kan opdage og dele services mellem mobil telefoner i et ad hoc netværk. Platformens effektivitet afhænger af hvor mange deltagere. Antallet af deltagere betyder dog også at der vil kunne forekomme flere ønsker om at opdage services og dermed øge den samlede energi brugt i netværket. Til at demonstrere en sådan platform er der udviklet en prototype. Prototypen er udviklet på baggrund af en kravspecifikation, deployment og aktivitetsdiagrammer som er analyseret og designet ved hjælp af UML 2.0. Udvilklingsmiljøet er Symbian s60 til Nokia N95 8GB mobil telefoner. Programmeringssproget til udviklingen af prototypen er Python til s60.

Funktionaliteten af prototypen er verificeret i test kapitlet, derudover er der lavet en analyse af platformen der giver en ide om hvordan det implementeret software på mobiltelefonerne yder med hensyn til energiforbrug med henholdsvis reaktiv eller proaktiv kommunikation.

Platform prototypen opfylder de krav som er stillet og det er muligt ved hjælp af platformen at opdage og distribuere services.

Preface

This report is developed as a final project in computer engineering with a specialization in communication systems at the department of Electronic Systems at the University of Aalborg in the fall 2009. This report is intended for people with basic knowledge of service discovery applications programming.

This report documents analysis, design, implementation and product analysis of a service discovery platform which is also able to distribute services as an App store for mobile phones. The report also addresses the energy consumed(mAh) with different communication types. Throughout the report the basic concepts of UML 2.0 methods are used in the design chapter. For project management the members of the group behind this project has used SCRUM methods for planning the different phases of the project. To keep track of all the tasks decided through SCRUM a server with XPlanner plus is set up. There will not be written any thing about the SCRUM processes or the XPlanner in this report, but to the exam a presentation is made of the group members experience with SCRUM and XPlanner during this project.

Throughout this report when ever there is referenced to a mobile phone it is to be assumed that there is a user using the mobile phone, and when there is referenced to a user it is a user with a mobile phone. When a mobile phone is said to be "nearby" another mobile phone, it means that they are both in range of each other and can communicate in an ad hoc network. In this report a packet consists of and header and a XML body.

The contents of this report appeals to people who are interested in service discovery, distributed systems and mobile applications. The sources used in this report are marked with a number inside [ ]. The source can then be found in the literature chapter. Every section in the design chapter starts with an introduction regarding figures and or the content in the following sections.

Attached to this report is a CD containing:

root	Which contains all source code written by the group during the project.
Sources	Which contains all sources for the report.
Test files	Contains all the test files / energy consumption calculation files.
master.pdf	Is the report in pdf format.

Word abbreviation

Ad hoc network:	Mobile ad hoc networking allows users to exchange information in a wireless environment without the need for a fixed infrastructure [1].
Application :	An application is a program for a mobile phone made for a specific purpose.
Broadcast :	Broad casting is when a message is send with UDP to an entire network.
DHCP :	Dynamic Host Configuration Protocol.
DNS:	Domain Name System. This is used to replace the IP address with a name instead.
Feature :	A feature in this report is a combination of software and hardware. So a feature is software which makes it possible to use specific hardware.
GUI:	Graphical User Interface
HTTP :	Hypertext Transfer Protocol.
HTTPMU:	Hypertext Transfer Protocol Multi cast. This is used for multi casting HTTP messages.
HTTPU:	Hypertext Transfer Protocol Uni cast. This is used for uni casting HTTP messages.
IP:	Internet Protocol.
IPv4:	IPv4 is based on 32-bit addresses.
Multi cast :	Multi casting is when a message is send with UDP to a specific IP range.
Service :	A service makes a feature available for others to use.
TCP :	Transmission Control Protocol.
UDP :	User Datagram Protocol.
Uni cast :	Unicast is when a message is send with UDP to a specific IP address.
UI:	User Interface.
XML:	Extensible Markup Language. This is used to exchange data between computers, embedded systems and so on.

Contents

CHAPTER 1

Introduction

Mobile phones in Danish families has over the last fifteen years grown from 14% in 1994[2] to astonishing 98% in 2009[2] which shows that almost every family has a mobile phone today. This amount gives a good indication together with 93%[3] of the Danish population in 2008 used a mobile phone [3]. That is almost every Dane which has mobile phone and used it. With that amount of mobile phones and usage of them this creates a large consumer market for mobile phones alone. When we look closer at the consumers usage of the mobile phone, as shown in figure 1.1, it clearly shows that mobile phones are no longer just used for sending text messages and calling. Now the mobile phones are used for video, music and pictures, but also other things like reading mail, surfing the web and receiving digital news. This shows that there also is a big market for mobile phone applications.

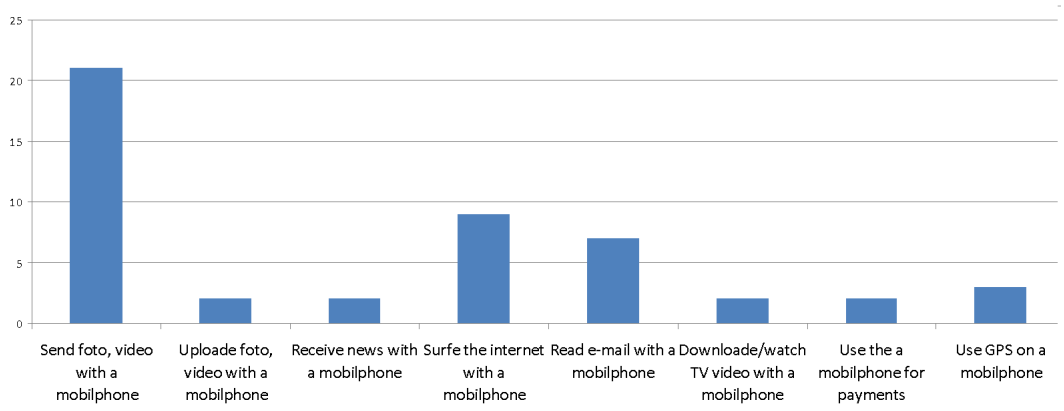

Figure 1.1: The consumers usage of mobile phone features [4]

As figure 1.2 shows the tree most used features are all together used equally from the age of 16 to 39 year old. It also shows that the usage of features is all together decreasing by the age groups above 40.

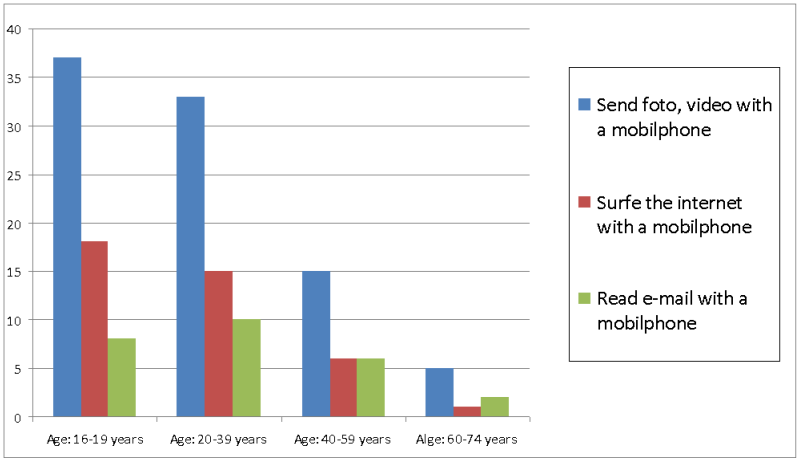

Figure 1.2: The tree most used mobile phone features divided into age groups [2]

For mobile phone features a new trend is begun with mobile operators, mobile manufactures and individual companies making app stores which are online stores where applications can be downloaded for free or by purchasing. They are all trying to make it as easy and user friendly as possible for the user to get new programs and thereby new features to their phone. This has been discussed in Harddisken, a Danish Radio program on P1 [5]. One of the more popular store is App Store [6], which is the name of the store, from the mobile manufacture Apple that supports the iPhone. This app store is fully controlled by Apple who decides which applications are available for costumers, which is done to ensure high user friendly applications. App Store offers over 50.000 applications and has untill now over 1 billion downloads since it was launched back in July 2008. Apple has a world wide marked share on 1-2% of the mobile phone market. Another popular store is Ovi Store [7], from the mobile manufacture Nokia that supports all Nokia mobile phones. This app store is not controlled in the same way as Apple do, but makes it possible to distribute applications for a wider range of mobile phone models as they have a world wide marked share of 33%. These application are of course offered on a world wide basic but they are also very popular in Denmark.

A common key factor for the popularity of these app stores is that it is easy to use for the ordinary user. Another side of these new features is that manufactures introduces new hardware capabilities in mobile phones, like touch screen display, GPS, compass, g accelerator and extended memory for upto 32GB, together with the app stores this gives the consumer new features. These app stores has created a new competitive parameter for mobile operators and manufactures in between. This is also discussed in previous mentioned radio program [5].

Together with app stores another new competitive parameter emerged, The Green Wave. This is everything regarding the environment in its whole life cycle from manufacturing, to transporting, to sales, to usage to reuse or termination of the product. Along with price and quality, app stores and The Green Wave are the future parameters for sale and marketing as discussed in the article "Grønne krav til elektronik vinder frem" [8]. Already now the green parameter plays a more effective role than the others do. In an analysis "Grøn it-bølge skyller ind over Danmark" shows that the companies not only think green because of the hype about it, but also because there is a lot of money to be saved in form of energy consumption and thereby reducing electrical bills and paying of CO_2 taxes.

When looking at mobile phones and The Green Wave Sony Ericsson has already produces their first environmental friendly mobile phone, Greenheart [9]. It distinguish itself by have a manual made of recycled paper, energy efficient display and waterborne paint meaning that the overall CO2 emissions of the phone are reduced by 15% compared to other mobile phones. Instead of producing new products, and waste around 20 to 50 million tons of electronic as the UN estimates [10], old products can be reused for other or new purposes in the terms of piggybacking. Piggybacking discussed in an episode of Harddisken [11] where the term is used together with electronic products. Here the idea is to take old technology, apply a new feature and reuse the product, for example taking and old mobile phone with infra red capabilities and make it into a remote control for the TV.

Maybe the term piggybacking could also cover not outdated hardware, but hardware which lacks a certain functionality, for example some mobile phones have WiFi but not access to a 3G net or limited access to the web. Would it then be possible to share the 3G bandwidth? Lets say that user 1 has access to a 3G net and user 2 does not, but user 2 is close to user 1. Would user 1 then be able to share its 3G net with user 2? Figure 1.3 is an illustration of the setup.

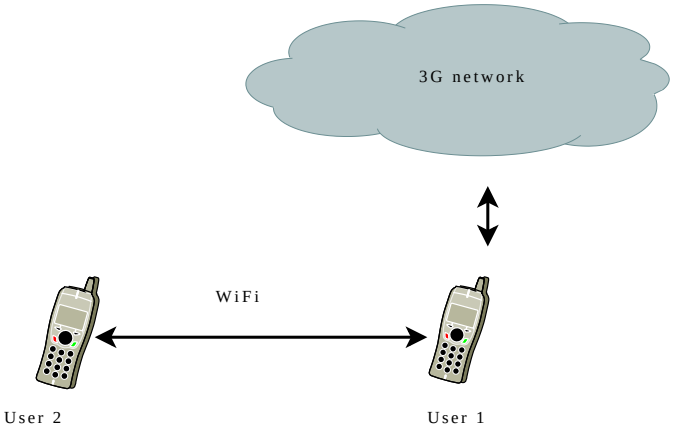

Figure 1.3: Bandwidth shareing

User 2 would then use User 1s 3G connection to connect to the web. Further more if there where more then one mobile nearby with 3G net would user 2 then be able to use both 3G connections and thereby increasing the bandwidth?

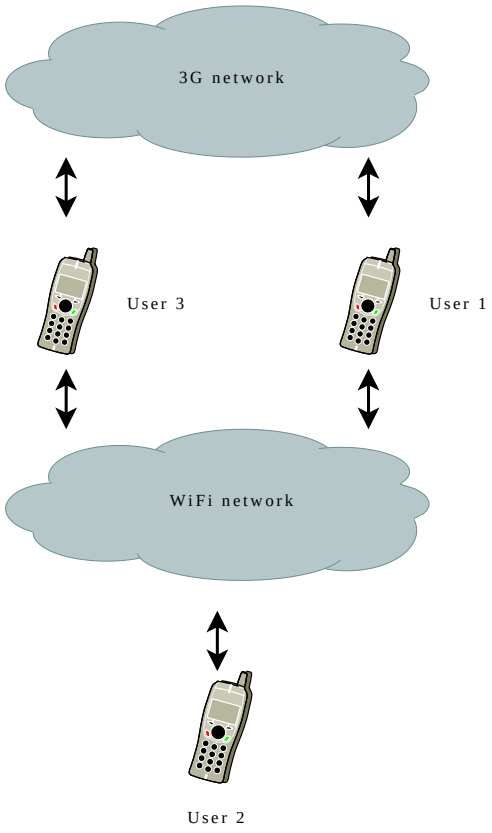

Figure 1.4: Using multiple 3G nets

On figure 1.4 there is a setup of a possible situation which adresses the issue of useing mutiple 3G nets.

Another idea could be if you imagen being able to listen to the same music as people near you. If some one has a new music album which you also like, and the person with the album would be able to let you listen to what the person is currently listening to. This could be somekind of boardcast like a radio or a service which is adverticed to other nearby mobile phones.

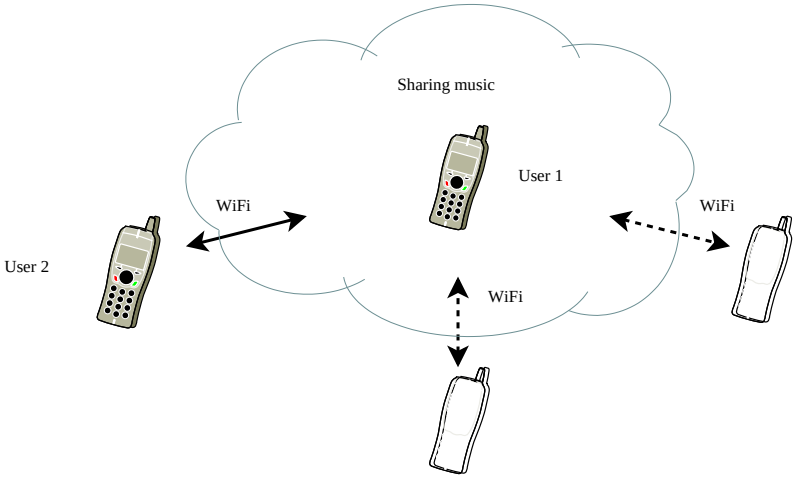

Figure 1.5: Sharing music

On figure 1.5 user 1 is listening to some music on the mobile phone. User 2 sees that user 1 is playing a number which user 2 also would like to listen to. User 2 is then able to connect to user 1's mobile phone and listen the same music. If one mobile can connect to another for the purpose of listening to music which is currently being played, why not being able to connect more then one mobile phone to user 1's mobile phone. When looking at figure 1.5 there are two faded out mobile phones, these two phones indicates that more then one mobile is able to connect to user 1's mobile phones and listen to the same music as user 1.

1.1 Initial problem

The introduction leads to the initial problem:

Is it possible to develop an universal application for a mobile phone which can discover and use other mobile phones services and features?

CHAPTER 2

System requirements

This chapter describes the requirements for the system that is designed and implemented. To have a real life scenario to base the system on a business concept is made which must meet specific requirements. First the business concept is described, then main components which makes it possible to develop a system that can support this business concept.

2.1 Business concept

The name of the business is WiFi3GShare, this business is a fictitious firm which the group of this project has made for the purpose of explaining the role of the service provider. WiFi3GShare provides a service called W3S which allows a user to share his/her 3G network connection with other nearby people who also has a subscription to WiFi3GShare. It is important to note that WiFi3GShare is not a 3G provider but only provides a service where users can share or use each others 3G net. W3S is shown on figure 2.1.

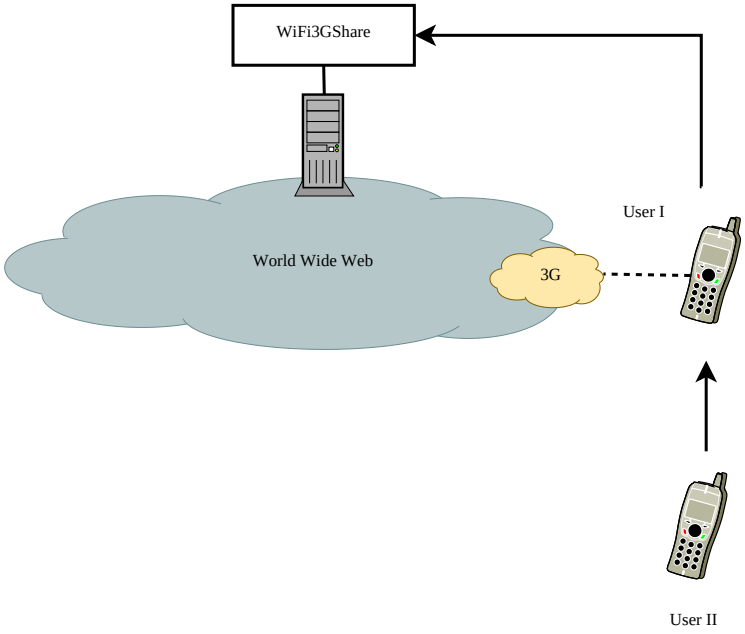

Figure 2.1: WiFi3GShare business concept

As shown on figure 2.1 User I has the W3S service on the mobile phone and has access to the 3G net, and user II has a subscription to WiFi3GShare, but does not have access to the 3G via the mobile phone. So to following happens before User II can use User Is W3S service.

1. First user II must discover that there is a mobile phone nearby which has the WiFi3GShare application installed.
2. User II must then asks User I for permission to use User IIs 3G net.
3. User II must then sends its ID to User I.
4. User I must forward the ID to the WiFi3GShare server to authenticate that User II has a subscription to WiFi3GShare.
5. The server must check the ID in the user database.
6. The server must reply to User I if User II should have access to the 3G net or not.
7. If User I gets a reply from the server saying that User II has a subscription User I allows User II to use User I's 3G net.

2.1.1 W3S

Why use W3S? Some Internet service provider has two kinds of subscriptions flatrate and flexrate [12]. Flatrate is where a fixed price is paid for the bandwidth regardless of usage and flexrate is where the amount of traffic to the Internet is paid for. These companies [12] offers a maximum of 10 gigabyte internet traffic. The idea with W3S you are able to share the 10 gigabyte internet traffic with people which have a subscription to the W3S service.

Example

Lets say that User I from figure 2.1 in average only uses 4,5 gigabyte per month of the 10 gigabyte User I has paid for. User I still has 5,5 gigabytes which goes to waste every month, why not have some one else pay for the 5,5 gigabyte or at least some of them.

In return

The W3S service allows User I to share its 3G net with others and then use up all the 10 gigabytes which are paid for. In return for every megabyte which User I allows other users to use User I gets 1 point per megabyte. User I can then exchange points to money which the user can pay the subscription for the 3G net with. If all the 10 gigabytes are used up and User I still wants to use the 3G net he can use his points by connecting to the 3G net through some one else's W3S service.

2.1.2 Subscription

The subscription for W3S is shown in figure 2.2. The account for W3S which is created on the WiFi3GShare server costs a one time amount of 50Dkr. If a user with an account wants to be able to use the W3S service it is possible to purchase 100 points for 20Dkr. For the users which share their 3G net and earn points are able to sell 100 points for 10Dkr.

	Account	Sell	Buy
W3S	50 Dkr	100 points for 10 Dkr	20 Dkr per 100 point

Figure 2.2: Subscription

2.1.3 How to get the service

WiFi3GShare distributes the service from a web page, the same web page as where it is possible to get a subscription. Further more if a user has the service installed the user should also be able to distribute the service it self. When looking at figure 2.1 User II should also be able to ask User I to get the service installed on User II's mobile phone.

2.2 The system

To support a business concept like WiFi3GShare there is needed a network where users are able to use each others services, also the network should be able to support that users enter or leave it randomly. There also needs to be some kind of platform where one user can discover other users which are sharing a service. The platform also needs to support distribution of services like an app store where it is possible to get an service installed from another user. And last the system need to support services like W3S.

2.2.1 Network

To support a network where people enter and leave randomly an ad hoc network is needed. As shown on figure 2.3 there are three different mobile phones which are in different ad hoc networks but are still able to communicate across these networks to each other, but not through one network to a third network.

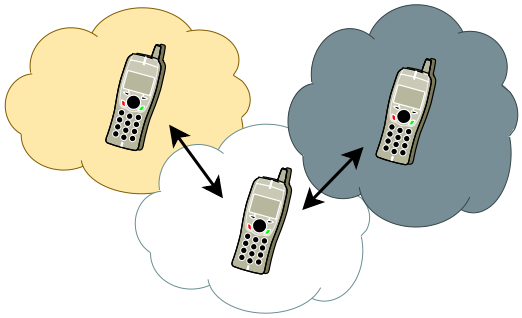

Figure 2.3: Ad hoc network

Network requirements	**Description**
1. Network type	The network must be of an ad hoc network type, because it supports the making of a network without any prior configuration or management and it supports users randomly entering and leaving the network.

Continued on Next Page...

Table 2.1 – Continued

Network requirements	Description
2. TCP/UDP	The network must support TCP/IP and UDP protocols, because they are a common way of making a network and many applications uses these protocols.
3. Communication	The communication for this network must be via WiFi. WiFi is chosen over Bluetooth because WiFi has a longer range then Bluetooth and a larger bandwidth up to 54Mbit. WiFi also supports TCP/IP and UDP protocols.
4. Distributed network	Each user must be able to automatic obtain an available IP address for making the mobile a part of an ad hoc network.

2.2.2 Platform

Based on the network 2.2.1 the platform must be able to discover other users on an ad hoc network. Also the platform should be able to distribute and install new services to support the spreading of services to ultimately get as many users as possible to share services.

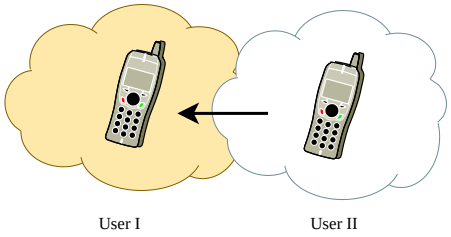

Figure 2.4: Discover a service

On figure 2.4 User II discovers User I which is sharing a service on an ad hoc network.

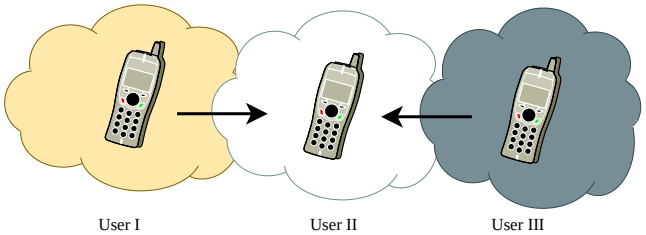

Figure 2.5: Distribute a service

On figure 2.5 User II gets a service from User I and now User III is able to see that User II is sharing a service.

Platform requirements	Description
1. Distributed server / client	The platform must able to communication as server / client where the client asks any available server which services they share.
2. Discover service	The platform must able to discover which services are available on the network.
3. Share service	Being able to tell which services are shared on a request.
4. Distribute service	If a service is shared, the user sharing it must also offer the software for using the service and further distributing the service. This software must be available to the user which wants to use the specific service.
5. Independent software	The platform must not be developed to some specific hardware, but suitable for all phones.
6. Default information	When sharing a service some default information must be available. These information contains IP of the host which is sharing the service, on which port the service is available and the name of service.
7. Start a service	Should be able to start a service based on IP address and port.
8. Manual installation	The platform should be manual installed by all users.
9. A service	A service must include a client for use of the service and the service it self.
10. UI	There must be a UI for the user to be able to search for services, share services and configure the services.

2.2.3 Service

W3S should be able to support ad hoc networks, also it should be able to support the way the platform discovers a service, shares a service and distributes a service. The W3S service should work like described in section 2.1. W3S should also support the following three scenarios shown on figure 2.6:

Scenario I : Here there is only one mobile phone sharing the W3S service which is used by a user which has a subscription to WiFi3GShare.

Scenario II : Here there is only one mobile phone sharing the W3S service but there are two users with a subscription to WiFi3GShare using the service.

Scenario III : Here there are two mobile phones sharing the W3S service and only one user with a subscription to WiFi3GShare using the service on both mobile phones.

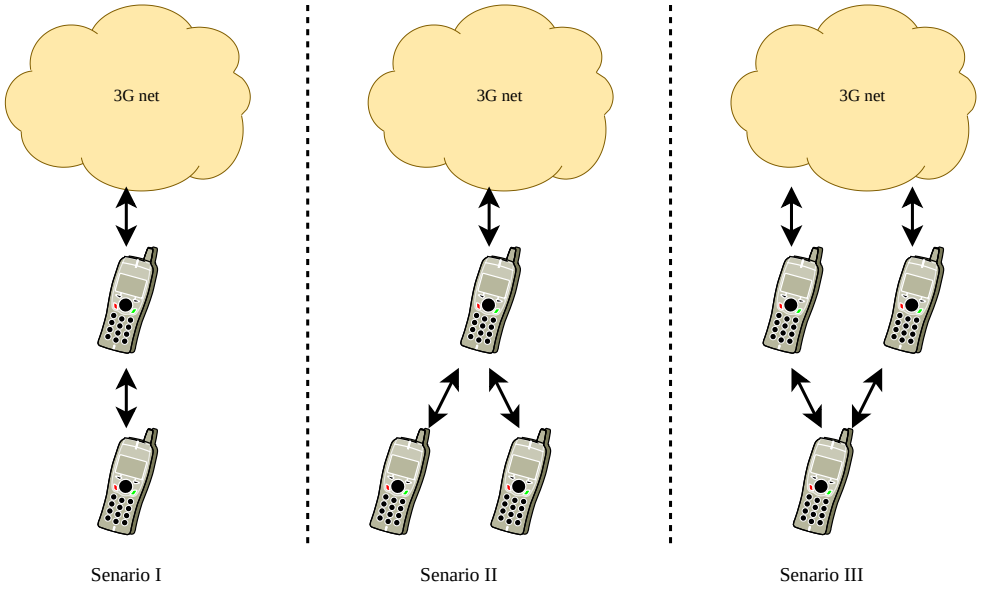

Figure 2.6: Three scenarios

W3S requirements	Description
1. Start service	W3S most be able to start on the parameters from the platform and determine if it should start as a use or share client.
2. Start page	When using the W3S the first web page which will be shown is the WiFi3GShare web page.
Support the platform	The W3S most support the platform which makes the basis for discovering, using and distributing the service.
3. Scenarios	The W3S most support all three scenarios from figure 2.6.
4. User account	The user most have an account at WiFi3GShare, the account should be created on the web page or from the start page of the W3S service.
5. Only share if	It should only be possible to start a W3S share client if the user has a 3G subscription from a third party (any 3G provideres are ok) and a mobile phone with WiFi capabilities.
6. Only use if	It should only be possible to use the W3S service if the user has a mobile phone with WiFi capabilities.
7. Monitor data traffic	W3S should be able to monitor all data which is shared to other users.
8. Configure W3S	It should be possible to configure W3S with a user ID and password to the WiFi3GShare account.
9. WiFi3GShare server	Should be able to communicate with the WiFi3GShare server to verify user accounts and send monitor data.

Server

The WiFi3GShare server should manage all accounts and host the default start page where is it possible to create, modify and show account data. Figure 2.7 shows the WiFi3GShare server.

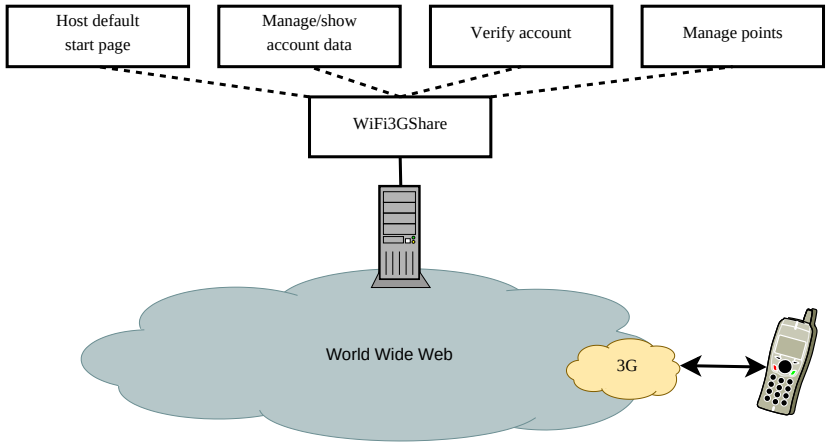

Figure 2.7: WiFi3GShare server

Server requirements	Description
1. Web portal	Host a web portal where it is possible to create a new user, manage existing user, view earned points and shown the default start page.
2. W3S requests	Verify user accounts and manage points.

2.3 Delimitations

Since this project is a short-term project it is necessary to make some delimitations. Also the reason for developing the product to a specific operating system called Symbian, is that the only mobile phones available for this project was Nokia N95 phones which only have Symbian as the operating system.

2.3.1 Delimit to:

Design	The platform will be designed and a design draft for the W3S service will be put into the appendix section.
Product	The product of this project will only be a prototype which can demonstrate the platform.
Mobile phone	Nokia 95 8GB mobile phone. The Nokia phones use SymbianOS which is an open source system. This makes it possible to make your own programs for the mobile phones with out the manufactures consent.
Symbian s60	The product will be developed to Symbian s60 OS.
Pys60	The software will be programmed in Python for s60.
Scenario I	Due to time concerns and that the product will only be able to demonstrate the platform it is decided to only implement scenario I from figure 2.6.

2.3.2 Delimit from:

Security	There will not be looked into any security issues (for example SSL for data connections and so on).
Real time	There will not be stated any real time demands for the product.
Error Handling	Because this the product is a prototype there will not be designed any error handling, but the implementation will contain the necessary error handling to get the product to work as intended.

CHAPTER 3

Existing platforms analysis

This chapter analyses the concepts of making a network, discover services and sharing services in the existing platforms, Universal Plug and Play, Zero configuration and Service Location Protocol. The platforms consist of protocols that are made for different layers in them. The analysed platforms are only considered from the decentralized system view, where the network communication is considered. The platforms are in the end compared to each other and the requirements for the system, to find the concept that matches the requirements best. This chapter also uses a lot of technical words which are explained in the word abbreviation part after the preface.

3.1 Principles used in the existing platforms

The existing platforms uses common principles in form of network communication and addressing types. The network communications type can be either proactive or reactive, which is explained here. When an ad-hoc network in the different platforms are created the addressing type used to get an IP adress is the Link-Local addressing method which is explained in this section.

3.1.1 Network communication types

The terminology proactive and reactive in communication types for networks covers how information is passed between nodes. As shown in figure 3.1 proactive communication is when a node advertise some information and does not wait for answers. This gives the problematic of when and how often this information must be sent. One way is to keep sending information even though this could mean unnecessary traffic if the information is not relevant for the receivers. Another way is to send the information when a new event happens, like entering a network and new informations are available, or a new state is reached. This requires to determine which events or states that pulls a new advertisement. Proactive communication is often used for registering process and information update. Reactive communication is when a node request some information and waits for response. This gives the problematic of more traffic for delivering the same information then proactive communication, but has the advantage of that traffic only is created when it is necessary compared to proactive communication. Reactive communication is often used when specific information is wanted.

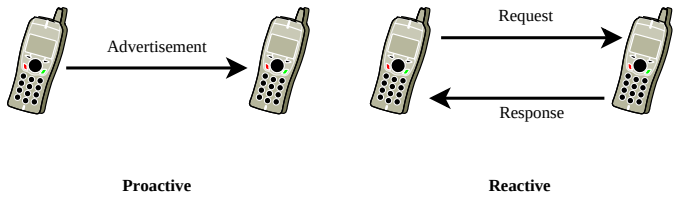

Figure 3.1: Proactive and reactive communication method.

3.1.2 Network addressing type

Network can be addressed by using the IP. IP is a protocol used for communicating data across a packet-switched internetwork using the TCP or UDP. The addressing assignment can be done static by for example a DHCP server or automatic by Link-Local addressing. Link-Local addressing is intended only for communications within one subnet of a local network or a point-to-point connection. They allow addressing IP to hosts without using a globally routable address prefix. This means that routers will not forward packets with link-local addresses. In Auto IP a network device picks a random address in the address range 169.254.0.0/16 and checks for its uniqueness. If the address already is in use by another network device, then it picks another address and tries again until it finds one. Auto IP is often used for network address configuration when no external source of network addressing information is available, for example from a DHCP server. It uses the range 169.254.0.0 to 169.254.255.255 where 169.254.255.255 is for multicasting.

3.1.3 Network transporting types

For transporting data over a IP based network TCP and UDP are common used protocols. They are both build on the principles of server client connection, where the server offers a client a connection. There can be multiple clients and / or servers but minimum one of each. When a connection is made, they can both send data to each other. The main difference between UDP and TCP is that TCP ensures that every package is received, where UDP does not have any. They both ensures correctness off each package. Because TCP have this insurance for delivering packages it uses more packages, and thereby also more time and energy, for delivering the same amount of data compared to UDP. This is also the reason for UDP can deliver more data on the same time compared to TCP. TCP is used where the guarantee of each packet reaches its destination is the most important, for example normal web browsing and home banking. UDP is used where speed is the most important, for example streaming.

3.2 Existing platforms

For designing a new platform for handling the Business Concept 2.1 and the requirements for the system 2.2, an analysis of existing platforms which are dealing with the issues of ad hoc network, service discovery and sharing of services on a local area network is made. The analysed protocols is considered from the angle of decentralized systems. This is because of the demand stated in section 2.2 that the platform has to have a distributed network. This analysis is to find the best suitable communication method for meeting the requirements.

3.2.1 Universal Plug and Play

Universal Plug and Play (UPnP) [13] is a platform of network module layers, as shown on figure 3.2, where it starts from the button with making a network and steps upwards through the layers until the final communication between the services.

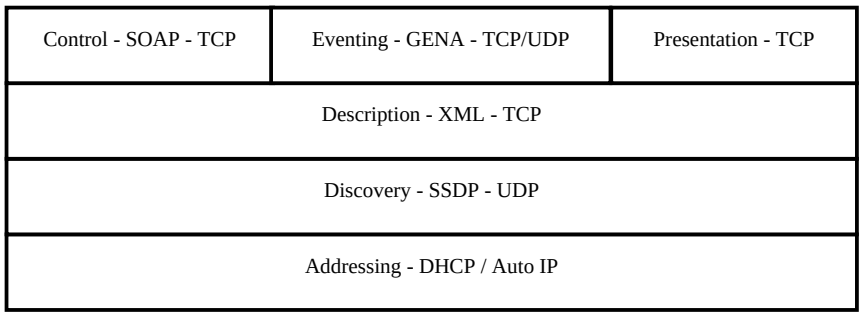

Figure 3.2: The UPnP platform.

The modules of the platforms works as followed.

Addressing - UPnP makes a network by first setting an IP address by trying to find a DHCP server and if it can not find one it uses Auto IP 3.1.2 to address one. If a DNS server is present on the networt UPnP registers its device name to it, and if not, the device uses its IP in the connections. Without any DNS server there can be multiple equal host names on the network.

Discovery - UPnP uses Simple Service Discovery Protocol (SSDP) for finding services available on the network over UDP and uses HTTP for communication. SSDP takes tree roles, advertising, searching and responding. Each device multicast advertises to all other devices when arriving or leaving the network and can renew its lease, all this it performed with HTTPMU. The lease time is an alive time set for each participant for when to recast a packet. SSDP multicast searches to other devices with the use of HTTPMU when one device looks for other devices or services. SSDP unicast a respond if a search message matches its criteria. The tree roles can all be implemented on every device depending on what role it needs to have on the network.

Description - UPnP sends a description in XML when asked for, containing device and service description over TCP and uses HTTP for communication. The service can contain one or more descriptions, control, eventing or presentation.

Control - UPnP uses control when actions is wanted to be performed on the device service. It uses Simple Object Access Protocol (SOAP) a framework for describing in XML what is in a message and how to process it. SOAP is used for represent remote procedure calls and responses in a decentralized, distributed environment over TCP using HTTP for communication.

Eventing - UPnP uses eventing when a state change is listen to from other devices. It uses General Event Notification Architecture (GENA) a framework for sending and receiv-

ing notifications in XML. SOAP is used for subscribe / unsubscribe to and notify devices over TCP/IP and administratively scoped multicast UDP using HTTP.

Presentation - UPnP uses presentation when a web site presentation of the device service is wanted. It uses a web browser over TCP using HTTP for communication.

There is two different roles in UPnP, a Control Point, usually the client, and a Device, usually the server, that contains the service. Both roles could be contained on a physical device. They both use the addressing layer to gain access to the network. The Control Point uses the Discovery layer UDP HTTPMU to find Services on Devices. The Device uses the Discovery layer UDP HTTPU to respond to Control Point and can also announce its presence or leaving via HTTPMU. Then the two roles makes TCP connections as server / client connections and uses the remaining layers. This gives a platform that uses reactive communication, by waiting for a respond, for everything except some parts of the Discovery where it uses proactive communication, by announcing its presence. The Platform gives the opportunity for using it both as a decentralized and centralized or combined system.

3.2.2 Zero configuration networking

Zero configuration networking [14] is a platform of network modules, as shown on figure 3.3, where it starts from the button with making a network and steps upwards until it has discovered services. After the service discovery an individual application must be made to communicate between devices.

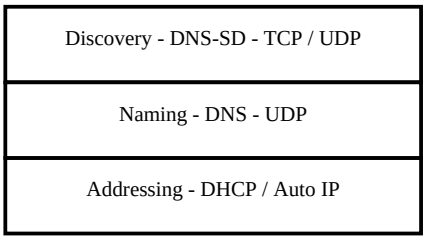

Figure 3.3: The zeroconf platform.

The modules of the platforms works as followed:

Addressing - Zeroconf makes a network by first setting an IP address by trying to find a DHCP server and if it can not find one it uses Auto IP 3.1.2 to address one.

Naming - Zeroconf depends on having a host name and uses a DNS server if it is present, if it is not present, it uses multiple DNS (mDNS). It is a distributed DNS server where every device has a register for linked IP addresses and host names. mDNS multicast advertise and requests to all other devices over UDP. When a host name must be found and a request is send to all devices, the device waits for a unicast reply, either from another device that tells what the host names IP is, or from the device that consist of the host name itself, all over UDP. mDNS advertise itself when a device arrives or leaves the network and can renew its lease. The lease time is an alive time set for each participant for when to recast a packet. The packet structure used by mDNS is unique for the system.

Discovery - Zeroconf uses DNS Service Discovery (DNS-SD) for finding services available on the network over UDP using mDNS or Dynamic DNS if a server is present. DNS-SD takes two roles, advertising and browsing. A device multicast advertises to all other devices when arriving or leaving the network and can renew its lease. Advertising is done using Digital Audio Access Protocol (DAAP)a communication that defines what services available is and how to use it over HTTP using TCP. A device browses for services on the network when it wants to use one. Browsing is done using Service records (SRV) a communication that specifies information on available services and Zeroconf SRV also specifies which one is used using TCP or UDP depending on the service. The two roles are meant to be implemented on every device.

Zeroconf builds on many of the used principles in DNS but does not contain definition of roles like UPnP. There is a client and server part, the server part be centralized or decentralised. The devices uses UDP for naming them self and then connects to each other using TCP or UDP depending on the service. This gives a platform that uses reactive communication, by waiting for a respond, for everything, except some parts of the Discovery where it uses proactive communication, by announcing its presence.

3.2.3 Service Location Protocol

Service Location Protocol (SLP) [15] is only a service discover protocol and not a complete platform. SLP allows devices to find services in a network mostly over UDP or TCP for larger messages. SLP takes tree roles, User Agents (UA), devices that search for services, Service Agents (SA), devices that announce one or more services and Directory Agents (DA), are devices that cache services.

It is not necessary to have a DA in the network for having a working SLP. The operation of SLP differs considerably, depending on whether a Directory Agent (DA) is in the network or not. When a device first joins a network it multicast a query for DAs on the network. If no DA answers it will assume that it is in a network without DAs. It is also possible to add DAs later, as they multicast a heartbeat packet in a predefined interval that will be received by all other devices. When an SA discovers a DA, it is required to register all services at the DA. When a device disappears the SA shall notify the DA to unregister it.

In order to send a request in a network without a DA, the UA sends a multicast UDP packet that contains the request. All SAs that contain matches will send a UDP answer to the UA. If the answer is too large to fit into a single UDP packet, a flag will be set and the UA is free to send the query directly to the SA using TCP, which can transmit packets of any size. In order to send a request in a network with a DA, the UA will send the request to the DA using either UDP or TCP. As every SA must register all services to the DA, the DA is able to fulfil the request completely and simply sends the result back to the UA. The UA and SA role are meant to be implemented on every device but the DA can also be implemented on every device, which would give a completely distributed system.

The request sent from the UA to the SA is a URL and can control the SA to perform commands. Depending on whether a DA is present the available URLs is registered to it or else directly to the SA.

SLP uses only proactive communication on the network by always waiting on a reply. The protocol gives the opportunity for using it as a decentralized or centralized depending on if there is a DA and that is centralized or decentralized.

3.3 OSI model comparison to the platforms

To have an idea of where the platform layers belongs compared to the OSI model, an illustration is made in figure 3.4. In the illustration it is also considered that the platforms are supposed to be implemented on the mobile phones with wireless LAN. As it shows, almost every difference lies in the application layer, except from Zeroconf that depends on a naming to its IP. This means that the different application modules easily can be implemented into each other by simple programming, because their underlying layers does not differs.

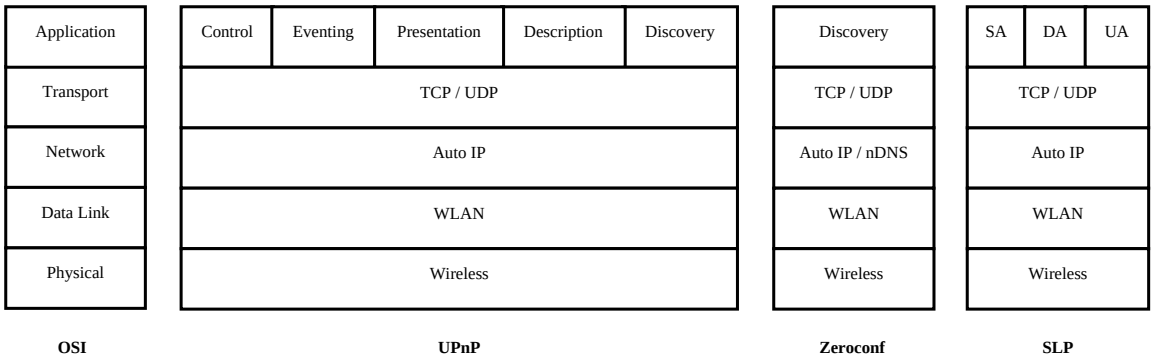

Figure 3.4: The OSI model comparison to platforms

3.4 System requirements comparison to the existing platforms

To determine which analysed platform that matched the system requirements 2.2 best, relevant requirements from networks requirements 2.2.1 and platform requirements 2.2.2 is matched up against the three analysed network platforms. The reference in the brackets refers to the specific requirement.

(Network 1.) - Network type - Both UPnP and Zeroconf makes use of Link-Local addressing which they use to manage the network. Zeroconf also needs to have a host name for having a working network, which gives some extra unnecessary implementation compared. SLP does not specify any network type.

(Network 2.) - TCP/UDP - They can all three supports TCP / UDP in their communication methods.

(Network 3.) - Distributed network - Both UPnP and Zeroconf makes use of Link-Local addressing that automatic assigns IP addresses. SLP does not consist of any.

(Platform 1.) - Distributed server / client - They can all three make a distributed server / client connection for communicating.

(Platform 2.) - Discover service - They can all three discover services, but have different ways of implementing the principle. There is one difference in the way UPnP can do the discover process compared to Zeroconf and SLP. Zeroconf and SLP need a distributed server for determine the participants on the network before any request for services can be made, where UPnP simply can wait until a request for a service is wanted and then ask the participants. The discover service requirement is only the part that ask

for the services. This is the Control point in UPnP, browsing in DNS-SD, Zeroconf and UA in SLP

(Platform 3.) - Share service - They can all three tell what service they have available and deals with the same different ways of implementing the principle like (Platform 2.) - Discover service. This requirement the roles is the part that replies on a request. This is the device in UPnP, advertising in DNS-SD, Zeroconf and SA and DA in SLP.

(Platform 4.) - Distribute service - None of the concepts are meant for distributing software for using a service, but only sharing commands for interacting with the service.

(Platform 5.) - Independent software - They are all three concepts and can be implemented on any phone having a TCP/UDP interface.

(Platform 6.) - Default information - They can all three send information to each other but Zeroconf is meant to use host names instead of IP, why it can not send the IP.

(Platform 7.) - Start a service - Both UPnP and SLP can start a service on a given IP address and port but Zeroconf needs a host name.

The analysed platforms are set into a simple table in figure 3.5 where "+" means it meets the criteria, "-" means it does not meets the criteria, "N/A" means that there is no specification for dealing with the criteria and "(+)" means it is possible but needs extra implementation compared to the other platforms.

Requirement	UPnP	Zeroconf	SLP
(Network 1.) - Network type	+	(+)	N/A
(Network 2.) - TCP/UDP	+	+	+
(Network 3.) - Distributed network	+	+	N/A
(Platform 1.) - Distributed server / client	+	+	+
(Platform 2.) - Discover service	+	+	+
(Platform 3.) - Share service	+	+	+
(Platform 4.) - Distribute service	-	-	-
(Platform 5.) - Independent software	+	+	+
(Platform 6.) - Default information	+	-	+
(Platform 7.) - Start a service	+	-	+

Figure 3.5: Requirements vs. analysed platform

By only looking at the table it shows that UPnP meets all of the system requirements except "Distribute service". This means that such a module must be added to the UPnP to meet the requirements.

CHAPTER 4

Platform design

This chapter designs the platform which is one of the two software parts the system consist of. The other part of the system is the service part. The service design is only a design draft and can be viewed in appendix A. The deployment digram contains both the platform and the service in order to give an overview of the complete system. In this chapter the UPnP platform layers is first considered for a design in the platform, where a specification of the concepts are made. Then two deployment diagrams are made, one for the platform and one for the service. The deployments diagrams and following design are made based on the UML 2.0. From the nodes in the deployment diagrams use cases are made. For each use case an activity diagram is made. Last a class diagram is made to show threads and classes needed for the implementation.

4.1 Platform layers

To determine the final platform specification for the design, the UPnP needs to be concretized in its specification and a module for distributing the service is needed to be added. The specifications for the platform layers are done by adding necessary specification to meet the system requirements 2.2 with the UPnP specification 3.2.1 and not include unnecessary implementation from it. The specification of the UPnP layers starts from the bottom layer.

4.1.1 Addressing

This system needs to have a self managed ad hoc network for the platform and service for communicate. Since both UPnP and Zeroconf makes use of Link-Local addressing this is used for manage the network.

4.1.2 Discovery

The system is a dynamic network in form of users moving around making clusters of ongoing ad hoc network, as explained in 2.2.1, where it is not possible to tell each cluster when leaving one and entering another on the same network. The presence of a user on a given time in a cluster is not relevant to the other users, whereas the services presence are. This means that advertising users presence between them, as SSDP advertising specifies, does not need to be done, but being able to always answer to a request for services on the network is. If the users where to advertise their presence on the network, to know which users available at a given time, an advertise message for a users presence should be sent with a high frequency, because of the mobile phones is able to move around. This would use a lot of resources for sending these advertisement and also create unnecessary traffic.

Only two roles are left in the SSDP, the searching and responding. The searching part is done in a

discover module, where it is possible to search for any services, and the responding part is done in a share module, that can reply on any request at any given time.

4.1.3 Description

The systems users does not know each others IP address, which is why it is not possible to make the description over TCP. Instead it is made with UDP as a part of a share module, where the service distribution also takes place instead of the Control, Eventing and Presentation layer. The description is still send in XML and contains information about what services available by which users.

4.1.4 Control - Eventing - Presentation

The system must from requirement 1. in platform 2.2.2 distribute the service as a server / client part, where all the communication meant for these three modules are done, which makes these three modules unnecessary. Instead a share module is made, where the distribution of the service takes place. This module must be able to distribute a service to other users.

4.1.5 UPnP Roles

The UPnP Roles are Control Point and the client they are done in a discover module and the device and server are done in a share module. They are to communicate over UDP HTTPMU and HTTPU and make a TCP connections as server / client connections for sending the service.

4.1.6 Illustration of the platform layers

The platform layers compared to the OSI model and UPnP is shown on figure 4.1 and the platform layers are summarized afterwards. The figure shows how the UPnP is mapped over to a concretized specification for the platform.

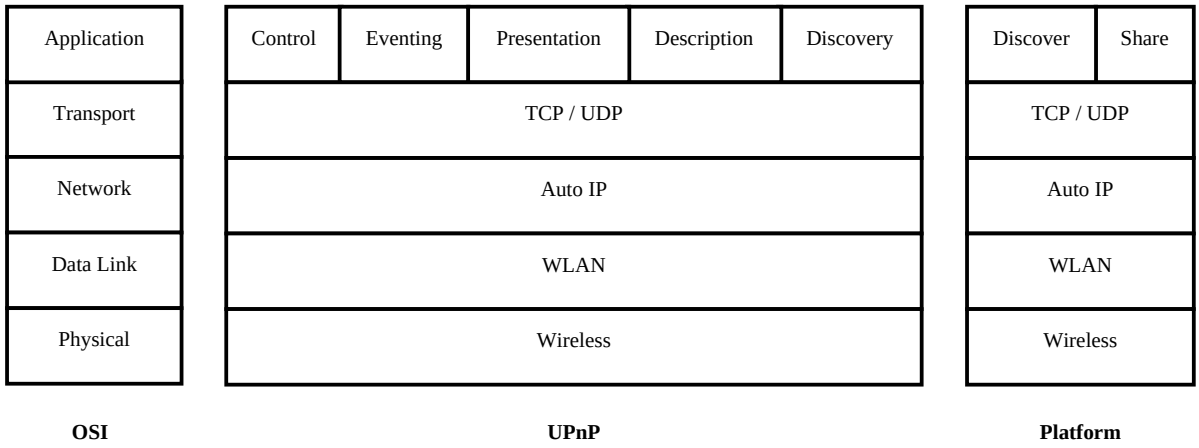

Figure 4.1: Layers in the platform.

Physical - Wireless - The network is communication through radio waves on the 2.4 GHz band.

Data Link - WLAN - The network is managed by WiFi IEEE 802.11 b/g.

Data Link - Auto IP - The IP on the network is managed by Link-Local addressing IPv4 RFC 3927.

Transport - TCP/UDP - The communication is going through UDP except when a service needs to be uploaded / downloaded, then the communication is going through TCP for maximum ensure the correctness of the data.

Application - Discovery - The Control Point role from UPnP is done here, where the searching for services over UDP HTTPMU and HTTPU is made. Also the download of services is done here.

Application - Share - The Device role from UPnP is done here, where the reply for searching for services over UDP HTTPMU and HTTPU is made. Also the upload of services is done here.

4.2 Deployment diagrams for the system

The purpose of the deployment diagram in the sense of UML is to model the overall software nodes needed in the system implementation. Deployment diagrams show the allocation of nodes according to the requirements made in chapter 2 and the application layer in the platform section 4.1. Figure 4.2 shows the different symbols used in the deployment diagram and an explanation for each of the symbols.

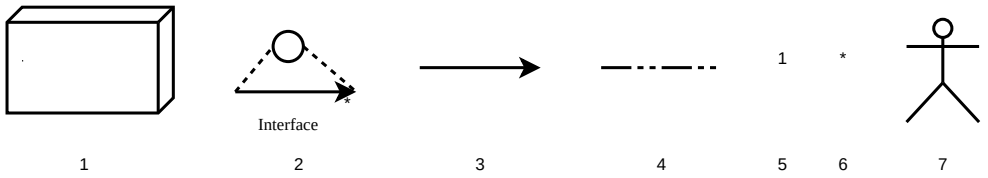

Figure 4.2: Explanation of the elements in the deployment diagrams.

1. The box symbolizes a node in UML context.
2. The special arrow connected with a dotted line to a circle symbolizes an interface between two nodes.
3. The arrow shows a communication between nodes, having the arrow pointing from the object that initialise the communication.
4. The line with two dots is to indicate and interface between two deployment diagrams.
5. The 1 is indicating that only one instance of a node can be started.
6. The * is indicating that endless instances of a node can be started.
7. The actor symbolises a user that interacts with a node.

There are made two deployment diagrams, one for the platform where the service discovery, installation of the services and service distribution is taken care of. The Platform is an universal software which make a basis for a network where services like W3S can communicate. The other deployment diagram is for the service where the W3S is used.

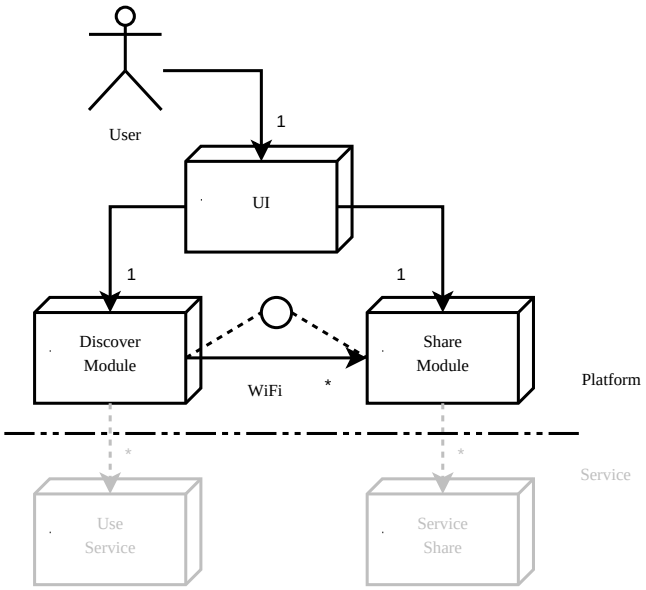

Figure 4.3: UML deployment diagram for the platform

This platform shown in deployment diagram 4.3 consist of a user interface that the user interacts with the system through. The user can start using a service or provide a service. These two parts are done in the Discover Module that discovers services and the Share Module that shares and distributes services. Afterwards W3S is started where it is possible to use the service or share the service shown in deployment digram 4.4.

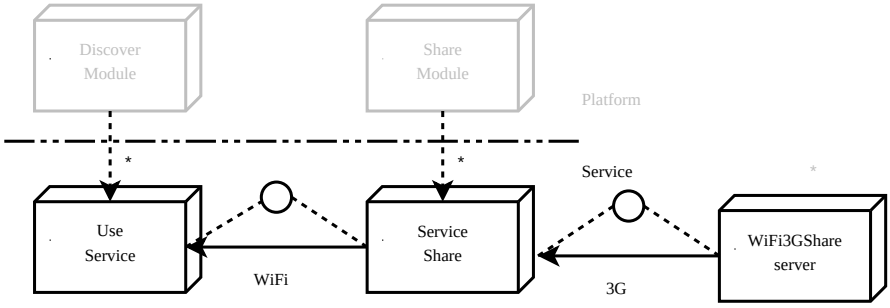

Figure 4.4: UML deployment diagram for the service

Figure 4.5 is to illustrate the communication between the nodes in both deployment diagrams.

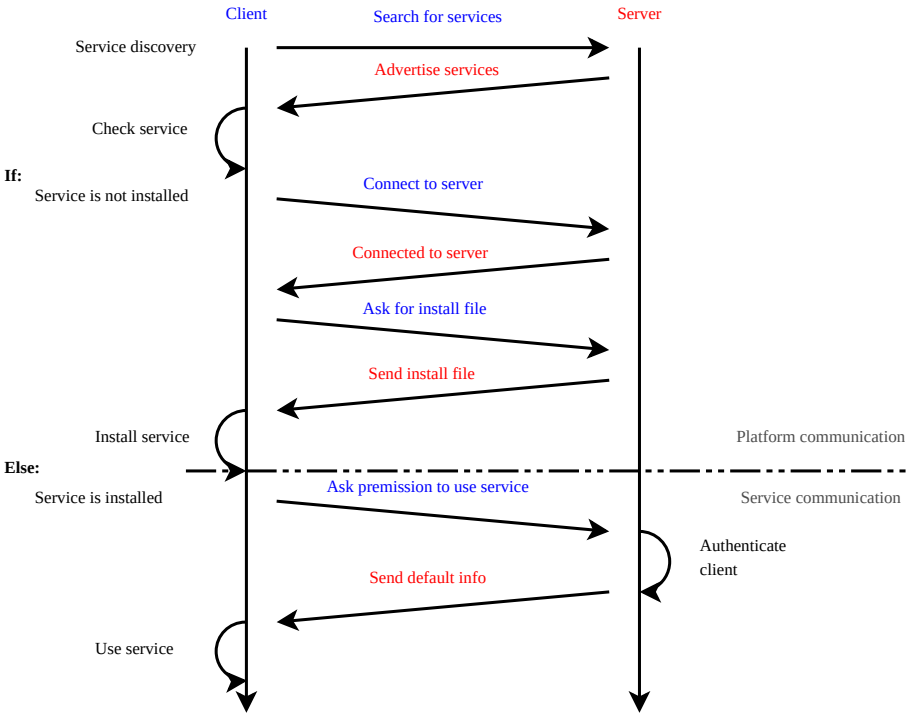

Figure 4.5: The communication between nodes

The client in Figure 4.5 has the role of the Discover Module in figure 4.3 and the Use Service in figure 4.4. The Server has the role of the Share Module in figure 4.3 and the Service Share in figure 4.4. The communication between nodes shows that first there is a discovery process to find out which services are available. When the available services are found the client checks if a specific service is installed, if not the service is installed. If/when the service is installed there is an authentication of the client if that goes well the client is allowed to use the service.

4.3 The platform UML design

From the nodes in the deployment diagram for the platform, the use cases and their activity diagrams are made. The purpose of the use cases in the sense of UML is to describe which actors can do witch functionalities in a node. The actors can be a direct interaction from a user or another nodes use case. Figure 4.7 shows the different symbols used in the use case and an explanation for each of the symbols.

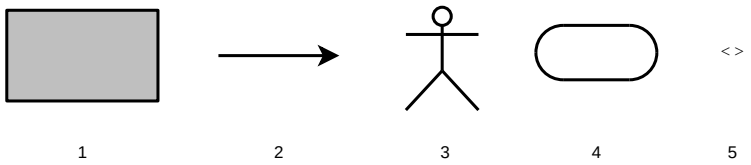

Figure 4.6: Explanation of the elements in the use cases diagrams.

1. The grey box shows a nodes collection of use cases.
2. The arrow shows a communication between an actor and a use case, having the arrow pointing from the object that initialise the communication. The object the arrow points from indicates an active object and making the other object a passive object.
3. The actor symbolises a user or another nodes use case that interacts with a use case.
4. The oval circle shows a use case.
5. Between the angled brackets the name of the node from the deployment diagrams is stated.

For each use case an activity diagram is made. Activity diagrams are a diagram technique showing work flows of stepwise activities, there all together shows the overall flow of control in a use case. The activities have one starting point, but can contain multiple end points. The activities are divided into boundaries that indicates what part of the program the activity are to be implemented in. Figure 4.6 shows the different symbols used in the activity diagram and an explanation for each of the symbols.

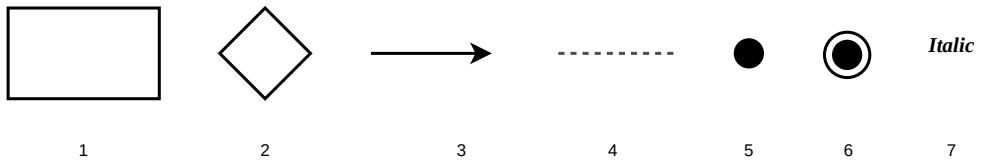

Figure 4.7: Explanation of the elements in the activity diagrams.

1. The box indicates an activity.
2. The diamond shows a state that must be evaluated before the following activity can be chosen.
3. The arrow shows a communication between activities, having the arrow pointing to the following activity.
4. The dotted line shows the line between the boundaries.
5. The bold dot indicates a starting point.
6. The bold dot with a circle around indicates an end point.
7. The italic words written in different colours describes the boundary.

4.3.1 The UI

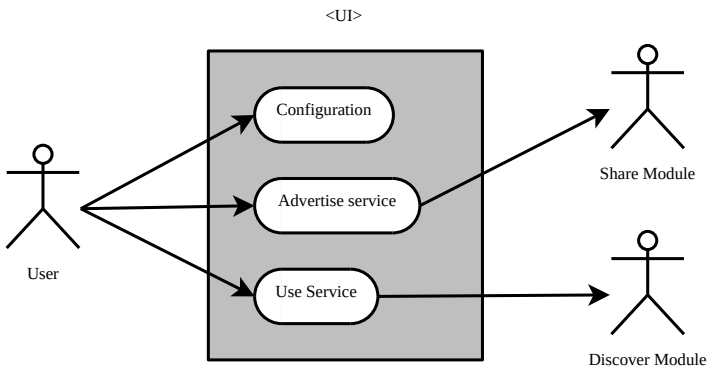

Figure 4.8: The UI use cases diagram

The UI node is where the user selects between configuring services, starts sharing services or search for available services. Figure 4.8 shows the use cases which are associated with the UI node from the deployment diagram on figure 4.4. The use case diagram has three actors, the user, Share Module and the Discovery Module. The user is the active actor which starts the communication, and the Share Module and Discover Module are the passive actors which only responds on a requested communication.

Configuration

The configuration is when the user wants to see which services the user can share or if the user wants to modify the configuration for a specific service. It is also possible for the user to delete an already installed service. First a list is shown for all the services which already are installed then the user must choose to modify the configuration for a service or delete it.

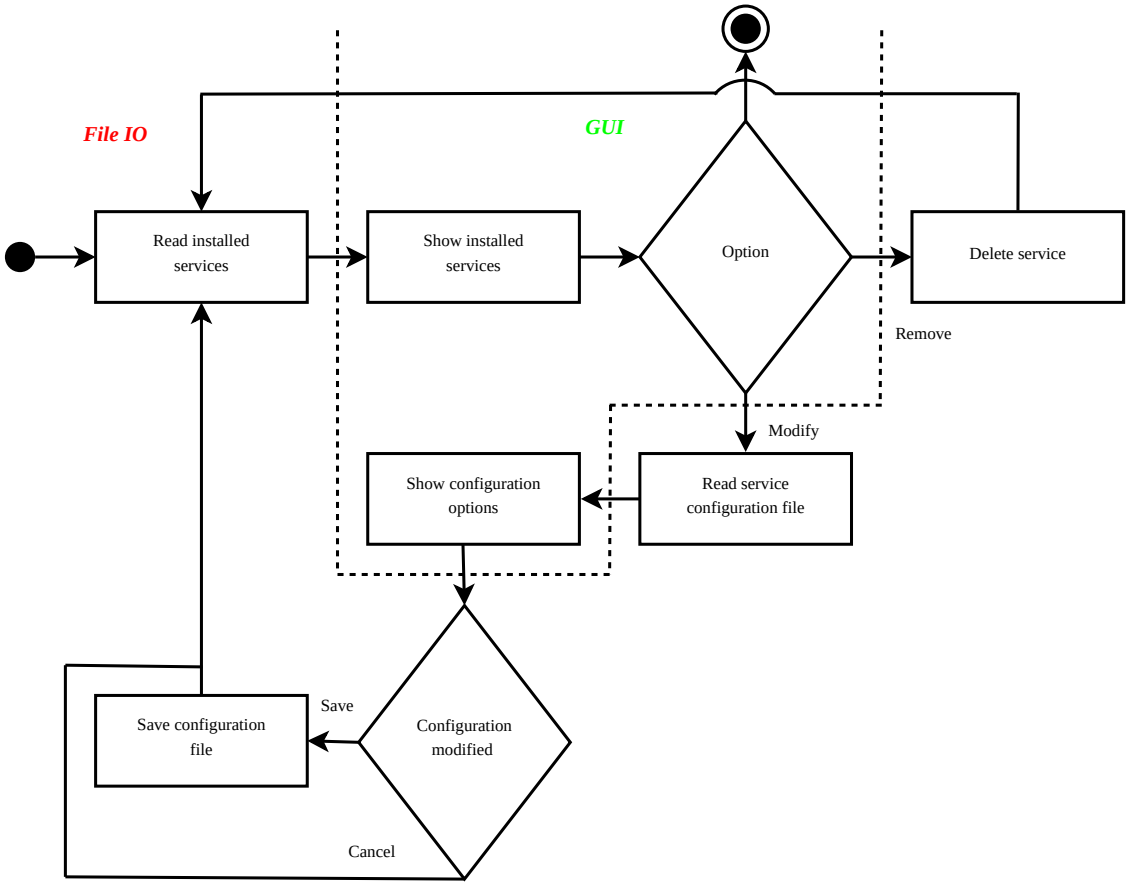

Figure 4.9: The configuration activity diagram

Figure 4.9 contains two boundaries, the File IO and GUI boundaries.

File IO - This takes care of reading in the already installed services, saving modifications to services and deleting services.

GUI - Shows the already installed services for the user, the option for deleting or modifying a service and the configuration options for a service.

The activity diagram has one endpoint. The endpoint is reached if the activity is completed successfully.

Advertise service

The advertise service is when the user wants to start sharing a service.

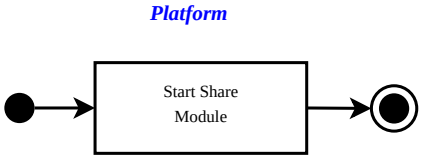

Figure 4.10: The advertise service activity diagram

Figure 4.10 contains only one boundary, the platform boundary.

Platform - Starts the Share Module.

The activity diagram has one endpoint. The endpoint is reached if the activity is completed successfully.

Use Service

The Use Service is when the user wants to search for available services.

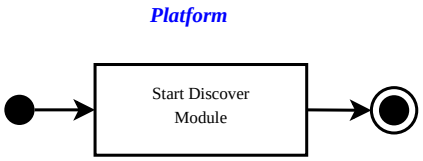

Figure 4.11: The use service activity diagram

Figure 4.11 contains only one boundary, the platform boundary.

Platform - Starts the Discover Module.

The activity diagram has one endpoint. The endpoint is reached if the activity is completed successfully.

4.3.2 The Discover Module

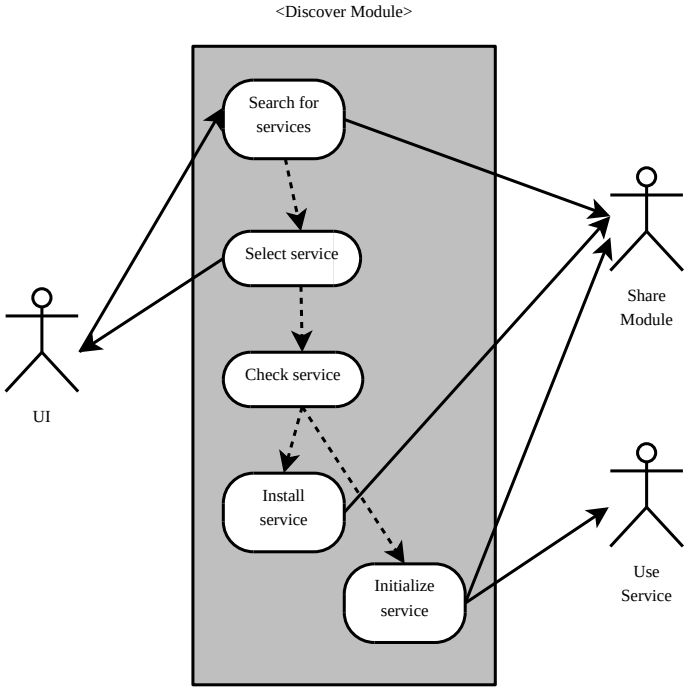

Figure 4.12: The Discover Module use cases diagrams

The Discover Module is when the user wants to search for available services. Figure 4.12 shows the use cases which are associated with the Discover Module node from the deployment diagram on figure 4.3. The use case diagram has two actors, the UI and the Share Module. The UI is the active actor which starts the communication and the Share Module is the passive actors which only responds on a requested communication.

Search for services

First a text box is shown, where a name of a specific service can be written. If nothing written it is a search for all services. Then all old available service information and the incoming buffer for packets are cleared, to make sure the user only get the current available services. Then a packet requesting available services is multicasted to all Share Modules. After the packet is multicasted the Search for services waits for responds, if there are any responds they are added to the service file.

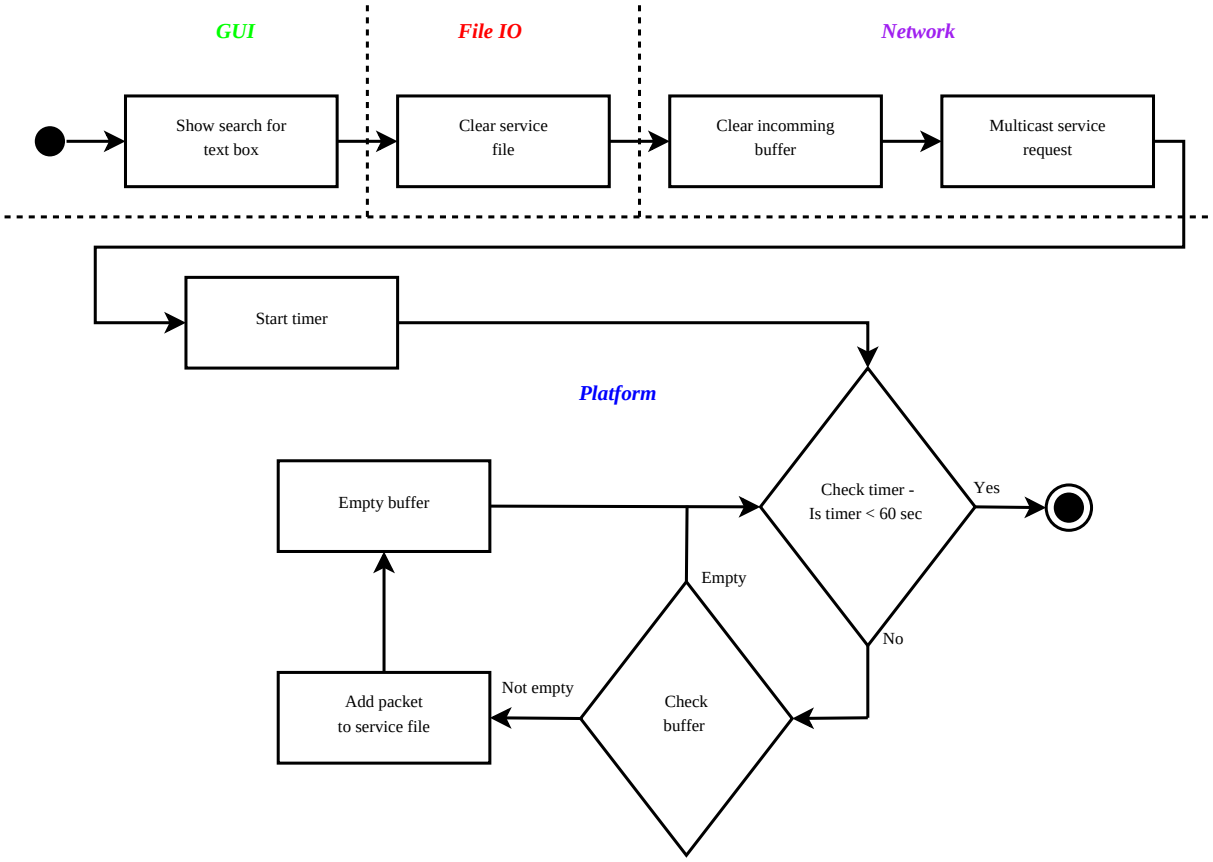

Figure 4.13: The search for service activity diagram

Figure 4.13 contains three boundaries, the File IO, Network and Platform boundaries.

GUI - Shows a text box where a service name can be written. No name is equal to any service.

File IO - Is for clearing the advertise file.

Network - The Network is for handling the multicast of the service request message.

Platform - This handles all the incoming packets and adds them to the service file and then deletes the packets in the incoming buffer.

The activity diagram has one endpoint. The endpoint is reached if the activity is completed successfully.

Select service

The select service is when the search for service is done and if there where found any available services they are displayed for the user. The user then selects which service to use.

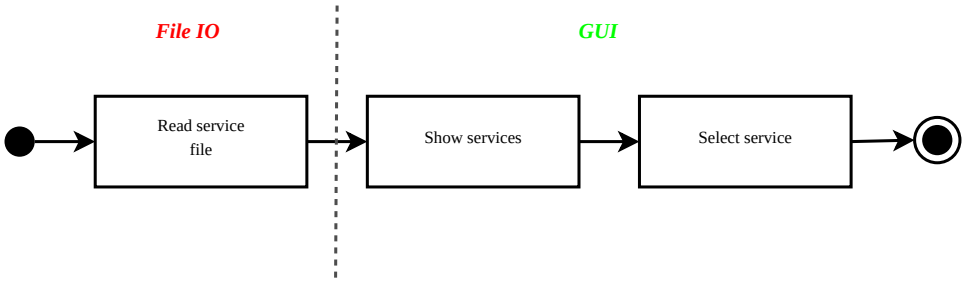

Figure 4.14: The select service activity diagram

Figure 4.14 contains two boundaries, the File IO and GUI boundaries.

File IO - File IO reads the service data base and saves it.

GUI - Shows the available services and returns the selection.

The activity diagram has one endpoint. The endpoint is reached if the activity is completed successfully.

Check service

The check service starts after a service is selected by the user. The service selected is then checked if it is already installed. If it is not installed the install service use case is started.

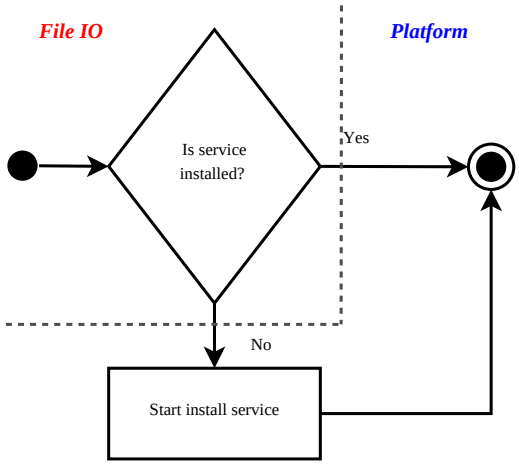

Figure 4.15: The check service activity diagram

Figure 4.14 contains two boundaries, the File IO and GUI boundaries.

File IO - File IO reads if the selected service is installed.

Platform - The platform starts the service install if the service is not installed.

The activity diagram has one endpoint. The endpoint is reached if the activity is completed successfully.

Install service

If a service is selected by the user and the check service did not find the service, the install service is started. The install service requests an install file and waits for a connection. When a connection is made, the install file is received and saved and the connection is closed. The the service configuration is started, where the user must enter basic informations for the specific service. These configurations can later be viewed and modified via the UI in the configuration 4.3.1.

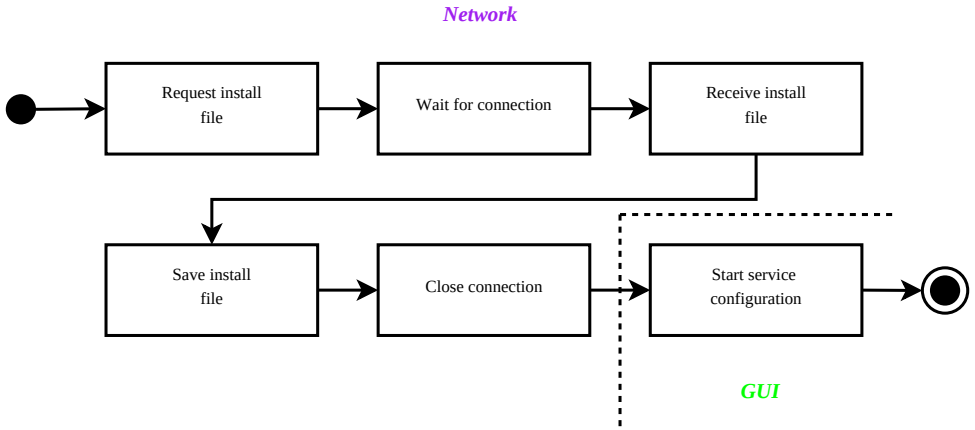

Figure 4.16: The install service activity diagram

Figure 4.16 contains two boundaries, the Network and GUI boundaries.

Network - The network requests a specific service, waits for the connection, receives the requested service, saves the file and closes the connection.

GUI - Starts the Configuration in UI 4.3.1 with the modification option.

The activity diagram has one endpoint. The endpoint is reached if the activity is completed successfully.

Initialize service

The initialize service is when a service is selected and ready for use. The initialize service then sends a service start request to the Share Module, waits to receive a reply containing the necessary information to start the service. The service needs to know three parameters to be started. It needs to know it should start as client, an the IP and port to connect to. These information are then passed on to the start service.

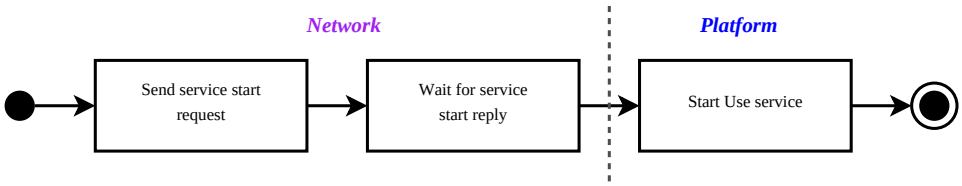

Figure 4.17: The initialize service activity diagram

Figure 4.17 contains two boundaries the Network and Platform boundaries.

Network - This sends the service start request and awaits a reply containing information on how to start the service.

Platform - This starts the service as a client for connection to an IP and port.

The activity diagram has one endpoint. The endpoint is reached if the activity is completed successfully.

4.3.3 The Share Module

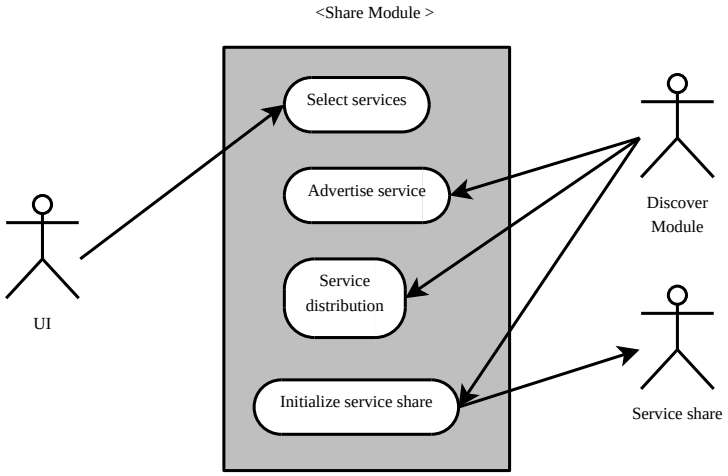

Figure 4.18: The Share Module use cases diagram

The Share Module is able to share and distribute the services which the user wants share the usage of and provide to others. Figure 4.18 shows the use cases which are associated with the Share Module node from the deployment diagram on figure 4.4. The use case diagram has three actors, the UI, Discover Module and the Service share. The UI and Discover Module are the active actors which starts the communication and the Service share is the passive actor which only responds on a requested communication.

Select services

The select services is started when the user wants to select which services to share. The select services read which services are installed, if any, and then shows a list where for the user to select which services to share. This list is then saved as an advertise list.

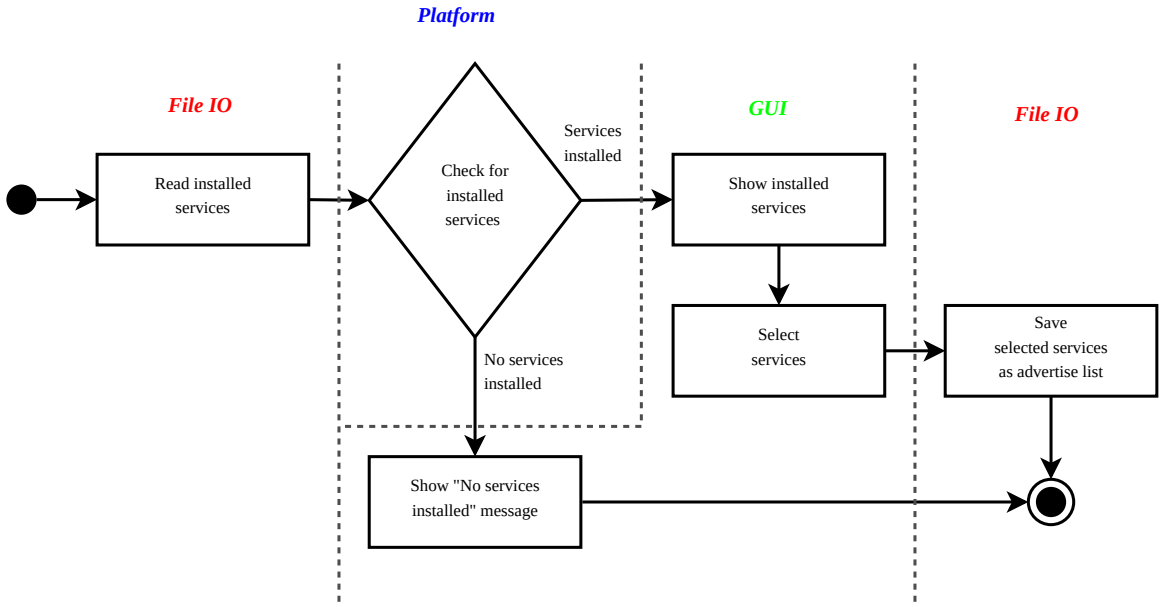

Figure 4.19: The select service use case

Figure 4.19 contains three boundaries the File IO, Platform and GUI boundaries.

File IO - Reads the installed services if any, and saves the advertise list.

Platform - checks whether any services are installed.

GUI - Shows the list of installed services or that no services are installed.

The activity diagram has one endpoint. The endpoint is reached if the activity is completed successfully.

Advertise services

The advertise services is when the Discover Module multicast the service request. The reply for the service request is a packet containing the whole advertise list if no specific service is wanted. If a specific request made and does not match any service in the advertise list, no reply is sent. The advertise list is the one made in the Select services use case 4.3.3.

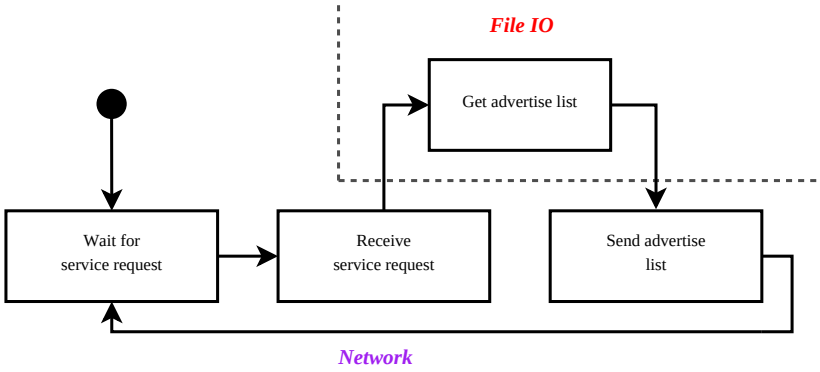

Figure 4.20: The advertise services use case

Figure 4.20 contains two boundaries the Network and the File IO boundaries.

Network - The Network receives the service request message compares the message with the advertise file and replies with a message containing all services in the advertise file, or if there where searched for a specific service it will reply if it has the service.

File IO - Reads the advertise list and clears the service request message.

The activity diagram has no endpoint because it is an endless loop. This also indicates that the use case must be implemented in its own thread.

Service distribution

When the Discover Module starts the install service activity 4.3.2 it sends a request for a install file to the service distribution. The service distribution then finds the install file, connects to the service requester, uploads the file, closes the connection and then wait for a new service request.

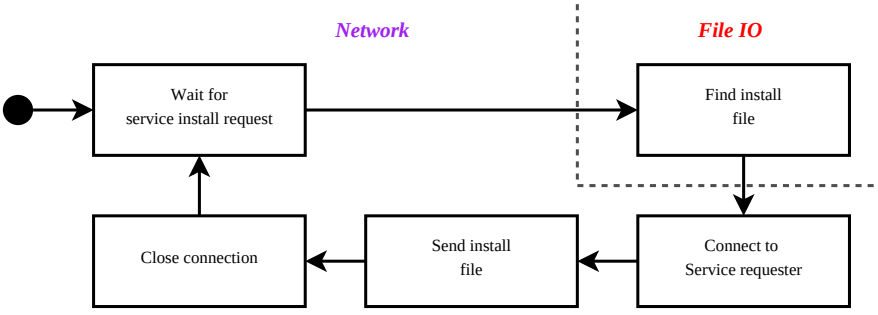

Figure 4.21: The service distribution use case

Figure 4.21 contains two boundaries the Network and the File IO boundaries.

Network - The Network waits for a service install request, then it makes a connection to the requester, sends the file and closes the connection.

File IO - Finds the requested install file.

The activity diagram has no endpoint because it is an endless loop. This also indicates that the use case must be implemented in its own thread.

Initialize service share

The initialize service share waits for a service start request received from the initialize service use case 4.3.2. then an available port for the specific service and the IP is found for starting the service. The last parameter the service needs to know, is that it should start as server. These parameters are then used to start the service. A service start reply is the send back to the service start requester.

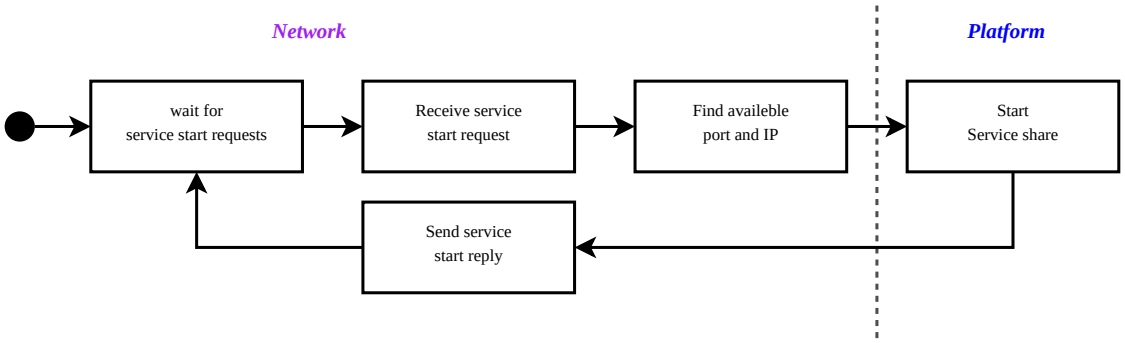

Figure 4.22: The initialize service share use case

Figure 4.22 contains three boundaries the Platform, Network and the GUI boundaries.

Network - The Network waits for a service request, and after receiving one, it binds the available port with a specific service and sends the IP and port back to the initiator. It also sends the IP and port to the requester.

Platform - Start the service as a server to the phones IP and an available port.

The activity diagram has no endpoint because it is an endless loop. This also indicates that the use case must be implemented in its own thread.

4.4 The platform class diagram

The purpose of the class diagram, shown on figure 4.24, is to show what classes needs to be made and how they interact with each other. For every node in the deployment diagram in the platform 4.3 and the boundaries in the activity diagrams from the platform UML design 4.3 classes are made. Three classes are also made for the three threads needed due to the three endless activity diagrams in the Share Module 4.3.3. Then the use cases are made into functions in the node classes. Figure 4.23 shows the different symbols used in the class diagram and an explanation for each of the symbols.

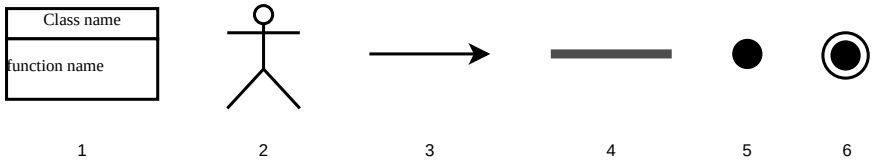

Figure 4.23: Explanation of the elements in the class diagram.

1. The box symbolizes a class containing functions.
2. The actor symbolises a user that interacts with a class.
3. The arrow shows an inheritance of another object, having the arrow pointing from the object that inherit the other objects functionalities.
4. The Bold line indicate a split up in threads.
5. The bold dot indicates a starting point in a main class.
6. The bold dot with a circle around indicates an end point.

The platform class diagram, shown on figure 4.24, shows that the program needs 3 threads except the main thread where the UI runs in. When the program is started, the user only interact through the UI class, where the end point also can be reached from. All together it gives ten classes to implement containing different functions.

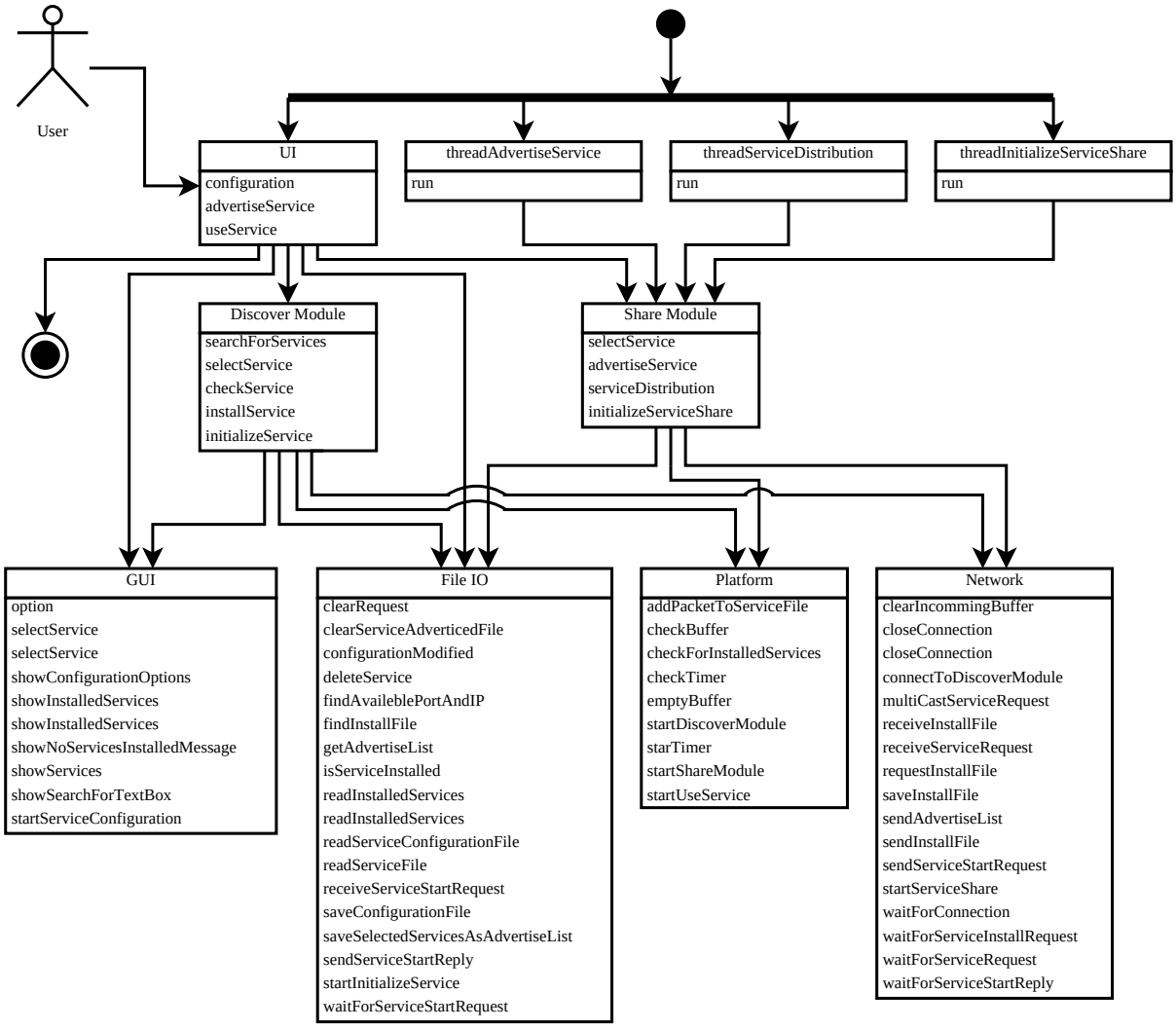

Figure 4.24: The platform class diagram

CHAPTER 5

Test specification

The tests in this project are based on blackbox tests of the use cases. Blackbox testing is a method of testing that verifies the individual use cases are working as designed. The test scenarios are described in this chapter. The data and the execution of the tests are carried out after the implementation in chapter 7.

5.1 The platform test

In the platform requirements in tabular 2.2 there is a requirement which states that the platform should be based on a distributed server / client relation for communicating between mobile phones. Based on the distributed server / client requirement the platform tests are split up into the two parts the Share Module and the Discover Module each part is tested separately. Both parts combined represents all the requirements from tabular 2.2 and their corresponding tests.

5.1.1 Discover Module tests

The Discover Module is the part that starts a search for available services and display them for the user so the user can chose which service he/she wants to use if any. The demands are listed in table 2.2.

UI:

1. Search for services:	The user can use a UI to search for a service.
2. Show available services:	When a search is performed then user should get all available services displayed.
3. Select available service:	Through the UI the user should be able to select any of the available services to use.

Discover service:

4. Search using the network:	When searching for services the Discover Module must use a network as described in tabular 2.1.
5. Receive search informations:	When receiving search informations this must be done on the same network as the search was send out on.

Distribute service:

6. Send service install request:	If a service is not install and the user wants to use it, the Discover Module must be able to send a service install request.

7. Receive service install file: The Discover Module must be able to receive the file from which a requested service can be installed.

Start a service:

8. Send service start request: When a service from the available services is selected and installed if needed the Discover Module must send a service start request.

Receive default information:

9. Receive default information: When receiving the default information these should be passed on to the service and the services should be started.

5.1.2 Share Module tests

The Share Module is the part which shares services, the demands is listed in table 2.2.

UI:

1. Advertise service: Through the UI the user must to able to select which services to advertise.

Share service:

2. Receive service request: The Share Module must be able to receive service requests on a network as described in tabular 2.1.

3. Reply on request: The Share Module must be able to reply the service requester with a message containing the which service are available.

Distribute service:

4. Receive service install request: The Share Module must be able to receive a service install request.

5. Send service install file: From the service install request the Share Module must be able to send the install file of the service requested to the requester.

Start a service:

6. Receive service start request: When receiving a service start request the Share Module must be able to start the service including the necessary interfaces if any.

Default information:

7. Send default information: When the service is started the Share Module must send the default to the requester containing the IP of the host which is sharing the service, the port the service is shared on and the name of the service.

CHAPTER 6

Implementation

The implementation chapter only contains a description of differences and clarifies some of the design, where all other implementation are to be considered implemented as designed. This chapter shows how the files for configuration settings and different packet types are implemented, also how the packets are stored inside the platform. There are examples of them. The chapter also describes how the different threads are implemented together with the intended files that contains temporary data. Further more, screen shots of the GUI are shown along with comments of how the user navigates through the GUI.

6.1 Platform files

The platform files the two files, the advertise list and Service configuration, the platform uses to store data in and all the incoming and outgoing packets.

6.1.1 The advertise list

The advertise list is implemented with the XML standards. The advertise list contains a list over the services the user is sharing through the Share Module. The advertise list is modified through the GUI, where it is possible for the user to select which services should be available. The advertise list looks like this:

```
1  <?xml version="1.0" ?>
2  <A>
3          <W3S/>
4          <SERVICE2/>
5          <SERVICE3/>
6          <SERVICE4/>
7          ...
8          ..
9          .
10 </A>
```

Code 6.1: Advertise list

Code example 6.1 shows the content of the advertise list which is stored in a file named AdvertiseList.xml.

6.1.2 Service configuration file

The service configuration file is, as well as the advertise list, implemented with XML standards. The service configuration file is also modified through the GUI. In the GUI the user gets displayed all the tags between the service start and end tags in this example the W3S start and end tag. The user then has the option to modify the data between the childtags start and end tags.

```
<?xml version="1.0" ?>
<W3S>
        <User> Test </User>
        <Password> Test1234 </Password>
        ...
        ..
        .
</W3S>
```

Code 6.2: Service configuration file

Code example 6.2 shows a service configuration file, in this case the W3S.xml file. The user then has an option to modify the "Test" and the "Test1234" as the user desires.

6.2 Packets

The packets are used to communicate between mobile phones. The packets are send via UDP and they consists of a header based on HTTP messages, and a XML body for the meta data. UPnP uses HTTP messages to communicate which is why it is chosen for the header of the packets in this project. The body which contains the data for the platform is based on XML instead of HTML, HTML is often used together with HTTP messages. The reason to chose XML instead, is that with XML it is possible to make your own tags, instead of using the fixed tags from the HTML standard.

When using XML there should be implemented a schema, which all who receive these packets should have in order to interpreted the XML body. In this case there is not implemented any schema, because the developers of this project has made their own XML parser for this purpose.

6.2.1 The header

The header is implemented based on specific parts of a HTTP message. The header can be either a GET or RESPONSE which determines if the packet is a request or a reply. If the header is a GET it is used to request data from others. If the header is a RESPONSE it is a reply to a GET packet. A GET packet header contains the host IP of whom has send the packet, the encoding used in the body and the size of the body.

```
GET
Host: 169.254.54.112
Content-Type: text/xml v1.0; charset Unicode
```

```
4 Accept−Range: Bytes
5 Content−Lenght: 200
```

Code 6.3: GET Packet header

Code example 6.3 is the header for a GET request packet.

- The first line indicates that this is a GET request which is send from the Discover Module to the Share Module.
- Line two is the host which has the IP address of the mobile phone which has send the packet. The reason to put the host IP address in the header, is due to that Pys60 does not support retrieving the IP address from a UDP communication when working with threads. So putting the host IP address in the packet solved this problem.
- Line three is the content type, this tells the receiver of the packet which encoding and charset is used in the body.
- Line four is the accept range is so the receivers know how calculate the length of the body.
- Line five is the length of the body.

```
1 RESP
2 Host: 169.254.55.113
3 Content−Type: text/xml v1.0; charset Unicode
4 Accept−Range: Bytes
5 Content−Lenght: 802
```

Code 6.4: RESPONSE Packet header

Code example 6.4 is a header for the RESPONSE packet made by the Share Module when a GET packet is received. The only difference in the two packets are the GET and RESP tag.

6.2.2 The body

To complete a packet both a header and a body is needed. The body is build based on XML standards. To pass the XML body the developers of this project has made their own XML parser because the version 1.4.5 of Pys60 did not support any XML parsers, this is only supported in a newer version 1.9.7 of Pys60. There are three different body types where each can be used with any of the two headers.

S - The S is for Service, this is used when a Discover Module wants to know what services the the Share Module offers.

SI - The SI is for Service Install, this is used when a Discover Module need to install a new service from the Share Module.

SS - The SS is for Service Start, this is used when a Discover Module has the service installed and need a Share Module to start the service.

```
1 <?xml version="1.0" ?>
2 <S/>
```

Code 6.5: S body for all available services

Code example 6.5 is for a GET S body which will make a Share Module reply with all the services the Share Module is sharing. This can also be referred to as a search for all available services.

If the user wants to search for a specific service the body would look like this:

```
1 <?xml version="1.0" ?>
2 <S>
3         <W3S/>
4 </S>
```

Code 6.6: S body a specific service

- Line one tells that this is a XML document based on XML version 1.0.
- Line two is the Service tag indicating that this is a S body.
- If the is a specific service it will be between the start and end S tag.
- Line four is the S end tag.

Code example 6.5 and 6.6 are send from the Discover Module. The reply for the body in code example 6.5 would look like this:

```
1  <?xml version="1.0" ?>
2  <S>
3          <W3S/>
4          <SERVICE2/>
5          <SERVICE3/>
6          <SERVICE4/>
7          ...
8          ..
9          .
10 </S>
```

Code 6.7: S body reply with all shared services

Code example 6.7 is the reply the Share Module makes when it has received a packet containing a body as shown in code example 6.5. In code 6.7 every tag between the S start and end tag are the services which this Share Module shares. If the Share Module had received a body for a specific service and the Share Module is sharing this service it replies with a packet of the same content as it received.

The complete packets

A complete S GET and S RESPONSE packet would look like this:

```
GET                                  | RESP
Host: 169.254.54.112                 | Host: 169.254.55.113
Content-Type: ...                    | Content-Type: ...
Accept-Range: Bytes                  | Accept-Range: Bytes
Content-Lenght: 200                  | Content-Lenght: 802
                                     |
<?xml version="1.0" ?>               | <?xml version="1.0" ?>
<S/>                                 | <S>
                                     |   <W3S/>
                                     |   <SERVICE2/>
                                     |   <SERVICE3/>
                                     |   <SERVICE4/>
                                     |   ...
                                     |   ..
                                     |   .
                                     | </S>
```

Code 6.8: S GET and S RESPONSE Packets

In code example 6.8 shows on the left side a S GET packet send from a Discover Module, the right side of 6.8 shows the reply made by the Share Module the S RESPONSE packet.

6.3 Threads and databases

Some modification to the The platform class diagram on figure 4.23 is necessary to complete the implementation. The modification is shown on figure 6.1 and the explanation the elements used is the same as in the platform class diagram and are explained in figure 4.23. The classes consisting of the boundaries are not changed, and therefore not included in this figure, to keep the figure more clear and simple to read.

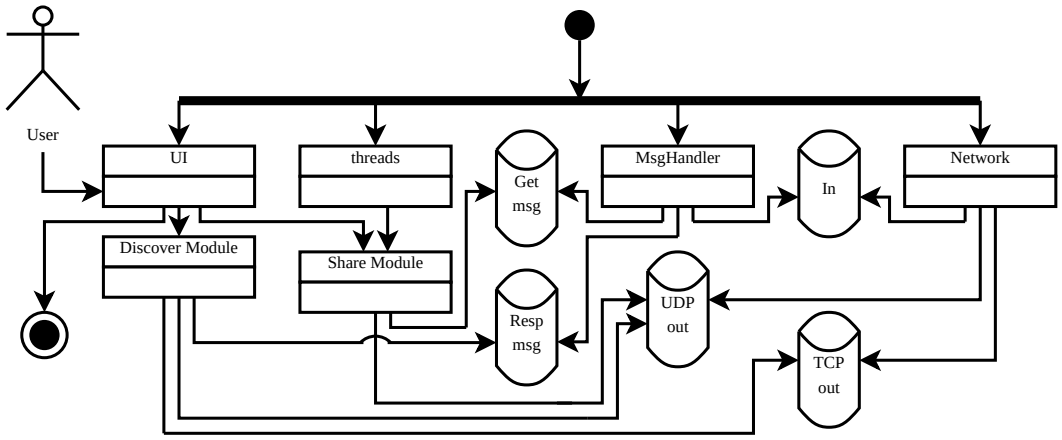

Figure 6.1: Databases in the platform.

Because Pys60 does not support sharing of native resources between threads, where sockets being one of these native resources, it is necessary to gather the network communication in one thread to preserve the designed functionalities from the class design. Because the network communication now is in one thread it is no longer necessary to keep functions like "threadAdvertiseService", "threadServiceDistribution" and "threadInitializeServiceShare" from The platform class diagram in separate threads. The reason they had a thread each was that they should be ready for incoming requests from the network. This is now done in the network thread. The former remaining functionalities from the three threads are now gathered in one thread called threads, where they run sequential after each other instead. The network part of the Discover Module is also moved to the network thread.

For communicating between the threads a simple lightweight database module called e32dbm is used, where one key is bounded to one value. The key is always the IP retrived from a packet. The value to a key is the packet which is received or about to be send. There are five different databases as shown on figure 6.1.

1. In - The In database is the databases which gets all packets directly from the network thread. From the database it is then analysed if the packet is a GET or a RESPONSE message.
2. Get msg - The GET msg database contains all the GET packets which are handled by the Share Module.
3. Resp msg - The Resp database contains all the RESPONSE packets which are handled by the Discover Module.
4. UDP out - The UDP out database are for all outgoing UDP packets.
5. TCP out - The TCP out database are for all outgoing TCP packets.

A thread is made called MsgHandler which takes care of moving packets from the IN database to the GET msg database or RESP msg database depending on if it is a GET packet or RESP packet. All UDP packets are put in the UDP out database and all the TCP packets are put in the TCP out database.

The rest of the platform design works as designed except the Discover Module and Share Module now communicates with the databases instead of directly to the other users via the network.

6.4 GUI

This section shows screen shots taken on one of the mobil phones which the developers of this project has borrowed from the University. The screen shots shows which options the user can chose from and how the user modifies service configurations, makes an advertise list and what happens when a user wants to search for available services.

Figure 6.2: Root menu.

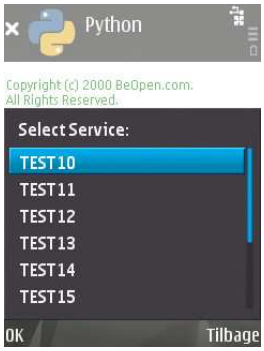

Figure 6.3: Configure a service.

Figure 6.4: Modify / delete menu.

Figure 6.5: Service options.

The root menu - The root menu from figure 6.2 is the first menu the user is presented with, from here it is possible to navigate to the service configuration menu, the advertise menu or if the user wants to use a service.

Configure service menu - The configuration menu leads to a list of all the installed services as shown on figure 6.3. Here is it possible for the user to chose which service needs to be modified.

Modify / delete menu - After the user has selected which service to modify, the user is then presented with a menu as shown on figure 6.4 where the user can chose to modify the service or delete it.

Service options - If the user chooses to modify a service a new list is presented to the user as shown on figure 6.5. These options are read from the service configuration file 6.1.2.

Figure 6.6: Modify value.

Figure 6.7: Save configuration.

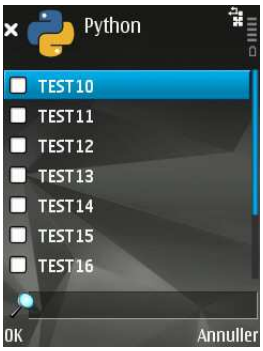

Figure 6.8: Advertise service.

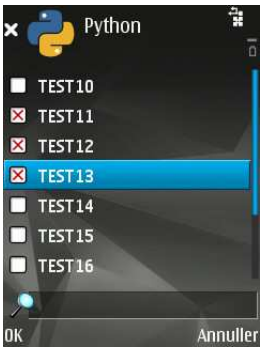

Figure 6.9: Select services to share.

Modify value - The user can chose to modify each of the values from figure 6.5. Figure 6.6 shows if the user had chosen to modify the value User. Here the user can enter a username for the selected service.

Save configuration - Figure 6.7 shows after the value is modified the user can chose to save the modification.

Advertise service - If a user chooses the advertise service from the root menu, shown in figure 6.2, the user is presented with a list of all the installed services as shown on figure 6.8.

Services to share - On figure 6.9 is shows the user selecting which services to share.

Figure 6.10: Use service.

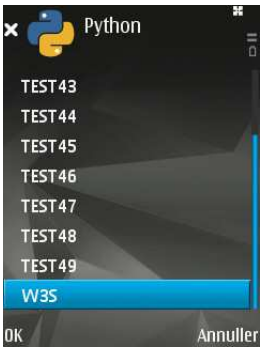

Figure 6.11: Service list.

Figure 6.12: Service server.

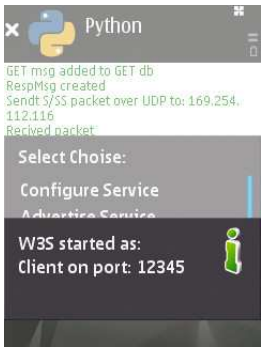

Figure 6.13: Service client.

Use service - If a user chooses the use service from the root menu, shown in figure 6.2, the user is presented for a text box as shown on figure 6.10. Here the user can type the name of a specific request wanted or leave it empty and get all services available.

Service list- After waiting for searching of services a list of available services is shown in a list as shown on figure 6.11. When a service is chosen it installs the service if it is not installed.

Service server and client - Since the service is not programmed, a simple pop-up, as shown on figure 6.12 and figure 6.13, is made to illustrate the service start and the parameters used.

CHAPTER 7

Accept test

7.1 Accept test

During the implementation of the individual functions and classes the source code consist of, the functions and classes tested as they were written. This is done to ensure they work as intended. The functions and classes tests are not documented in the report, but the blackbox tests from section 5.1 are performed with the following setup.

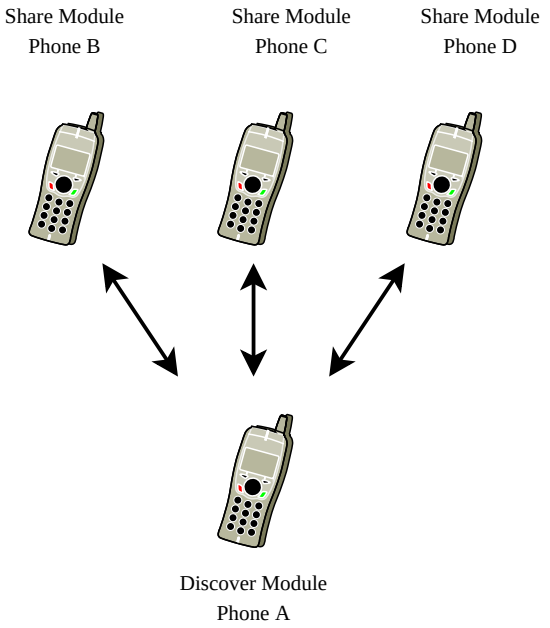

Figure 7.1: Accept test setup

Figure 7.1 shows the setup for the tests. There is one mobile phone (phone A) searching for services which will cover the Discover Module test, and the other three mobile phones (phones B,C and D) are the basis for the Share Module tests. In this test setup the phones B, C and D each have different services they share, all tests are performed for when phone A searches for all available services or for a specific service.

Discover Module tests

	Pass	Failed
1. Search for services	x	
2. Show available services	x	
3. Select available service	x	
4. Search using the network	x	
5. Receive search informations	x	
6. Send service install request	x	
7. Receive service install file	x	
8. Send service start request	x	
9. Receive default information	x	

Figure 7.2: Discover Module tests.

For the search for all available services the services from B,C and D where shown on mobile phone A. For the specific search A first searched for the service on B, then C and last D the specific services was found in each of the three cases.

Share Module tests

	Pass	Failed
1. Advertise service	x	
2. Receive service request	x	
3. Reply on request	x	
4. Receive service install request	x	
5. Send service install file	x	
6. Receive service start request	x	
7. Send default information	x	

Figure 7.3: Share Module tests.

Both phone B,C and D where able to receive the request for all available services, and they all replied A. In the case of the specific search the phone with the specific service was the only phone to reply to A.

CHAPTER 8

Platform analysis

Based on the implementation the following chapter is an analysis of the platform. This analysis is focusing on the energy consumed in different network communication types for service discovery to determining in which cases one type has a lower energy consumption than the other. The energy consumption considered is when looking at the energy consumed in the entire network over time. The calculations are based on an assumption of that the platform can support both a reactive and proactive network communication. This means the implemented platform processes of sending and receiving packets in a proactive network communication, are assumed to be the same for sending and receiving packets with reactive network communication. The only deviation is the reactive communication is user controlled and the proactive communication must be controlled by a frequency. There is discussed the different parameters which are possible to adjust, in order to achieve a certain performances of the platform. The performance is measured in energy consumption, which in this case is how much energy consumed from the mobile phone battery, where the lowest possible energy consumption is the best. Further more two scenarios with different network communication types and sizes are set up against each other.

8.1 Network communication types

With the platform implemented it is only possible to support reactive communication in the service discovery as described in 3.1.1. The reactive communication has two options which depends on the choices made by the user through the GUI. The second communication type, that could be possible, is the proactive communication which is discribed in 3.1.1, that is not depending of any choices made by the user. With both communication types all mobile phones are expected to have the platform installed so each of the mobile phones are participants in a network. Each time a users wants to discover any available services or a specific service on a network it is called an inquiry. In the reactive communication an inquiry is sending out a GET S request packet and the request packet is replied by other participants in the network. In the proactive communication an inquiry is an advertisement where the mobile phone sends out a multicast with all the services it shares.

8.1.1 Reactive communication

The reactive communication is event based, meaning that for each time a user makes an inquire the platform starts the Discover Module to send GET S request. When making an inquiry there are two options which the user can choose from, each of them has a different energy consumption.

Full search - The full search sends a GET S request packet to all participants except it self. With the full search all participants sends back a RESP S reply packet containing all the services from the advertise list the participant is sharing.

Concrete search - The concrete search is when the user is searching for a specific service, in this case the GET S request packet is send to all participants except it self, but only the participants which are sharing the concrete service which is searched for, will send back a RESP S replay packet saying they are sharing the service.

8.1.2 Proactive communication

In a proactive communication each participant sends, with a predefined frequency, a multicast message to all other participants on the network. The packet contains all the service the participant is sharing, looking like a full RESP S reply. This means, with the proactive communication, the user does not need to search for any services, but the user just need to choose which service to use.

8.2 Sending packets

When sending packets between Discover Module and Share Modules it has been observed, during test and implementation, that the packet size does not vary much and has little influence on the energy consumed on the network. Instead what does make a difference in energy consumed compared to the different packet types sent, is how many processes they are going through. This differs whether it is a request for all services or for a specific service. Figure 8.1 shows on the left side of the vertical dotted line, the platform sending a GET S packet, and on the right side of the dotted line the platform receiving the GET S request packet.

Figure 8.1 shows different coloured areas which each represents processes on the platform except the greyed out areas they are inactive processes.

Red - The red circled area represents the process on the platform which sends a GET S request packet. This process is performed the same way every time.

Green - The green circled area represents the process on the platform which receives a reply packet from the Share Module. This process is performed the same way every time.

Blue - The blue circled area represents the process on the platform where a Share Module receives a GET S request packet from a Discover Module. This process is performed the same way every time.

Purple - The purple circled area represents the process on the platform where the Share Module analysis the packet received and determines which sort of reply to make, if any services are available. This process is not performed the same way every time. Depending on the packet received, the Share Module determines if it is a request for all services or for a specific service. If the packet is a request for all the services, the Share Module simply replies with all the services it shares. If the packet is a request for a specific service, a comparison with the advertise list is performed, to see if the Share Module is to sharing this service, and it then makes a reply.

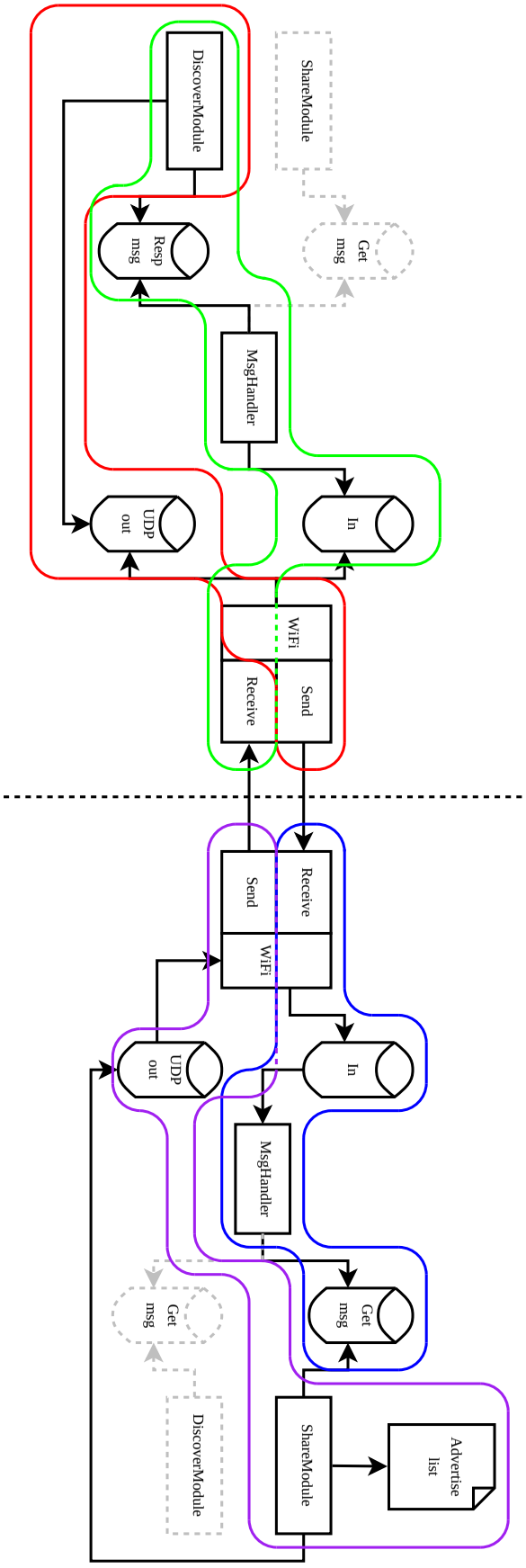

Figure 8.1: The path a S packet must through on the platform.

8.3 Energy Calculations

The calculation for energy consumed with the reactive communication, is divided into full search and concrete search. When calculating the energy following variables are used:

X	X is the participant sending the service discovery request and receiving the answers.
Y	Y is receiving a service discovery request but not answering with a reply.
Z	Z is receiving a service discovery request and answering with a reply.

Table 8.1: Participant types in a network.

Table 8.1 shows the three different participant types there can be in a network.

sFRQ	Energy consumed in mAh by a send full request packet.
sFRP	Energy consumed in mAh by a send full request reply packet.
sCRQ	Energy consumed in mAh by a send concrete request packet.
sCRP	Energy consumed in mAh by a send concrete reply packet.
rFRQ	Energy consumed in mAh by a receive full request packet.
rFRP	Energy consumed in mAh by a receive full request reply packet.
rCRQ	Energy consumed in mAh by a receive concrete request packet.
rCRP	Energy consumed in mAh by a receive concrete reply packet.

Table 8.2: Energy consumed by sending or receiving different packets in a network.

Table 8.2 shows a list of all the combinations of energy consumptions related to sending and receiving packets in a network.

E	is the total energy consumed in mAh.
p	is the number of participants in the network.
pr	is the number of participant replies in the network.
I	number of inquiries.
H	is the frequency of inquiries send over time.

Table 8.3: Network variables.

Table 8.3 shows the variables which are related to the network.

Full search with reactive communication

The full search with the reactive communication is shown on figure 8.2 where each square represents a mobile phone and each arrow between the mobile phones represents one communication. The notation X_f, Y_f, Z_f and $E_f$is to indicate that it is for the full search.

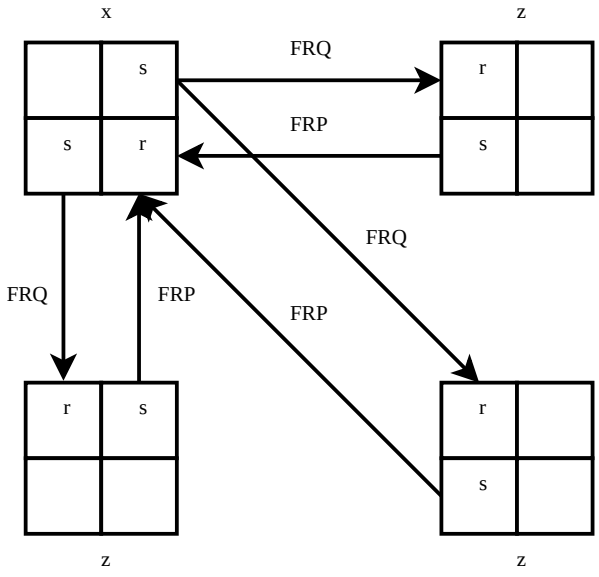

Figure 8.2: Full search with reactive communication

First the energy consumed for each participant is calculated. In the figure 8.2 it shows that there is only X and Z participants.

x : $X_f = sFRQ + pr * rFRP$

Participant x consumes the energy corresponding to the energy used by sending a FRQ plus the energy used on receiving a FRP from each participant replying.

z : $Z_f = rFRQ + sFRP$

Participant z consumes the energy corresponding to the energy consumed by receiving a FRQ plus the energy consumed on sending a FRP.

E : $E_f = I * (X_f + pr * Z_f)$

The total energy consumed by a full search with reactive communication is equal to each inquiry multiplied with the energy consumed by sending the inquiry (X), plus the energy consumed by receiving the request packet and sending the reply packet (Z) for each participant that replies (pr), which are all minus yourself (p-1).

Concrete search with reactive communication

The concrete search with the reactive communication is shown on figure 8.3 where each square represents a mobile phone and each arrow between the mobile phones represents one communication. The notation X_c, Y_c, Z_c and E_c is to indicate that it is for the concrete search.

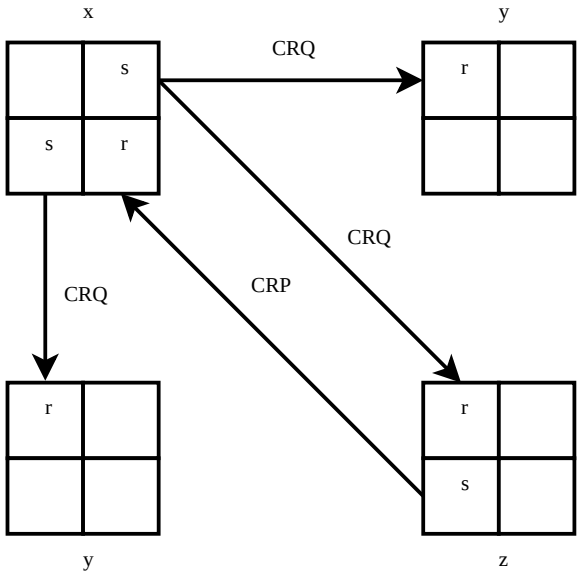

Figure 8.3: Concrete search with reactive communication

First the energy consumed for each participant is calculated. In the figure 8.3 it shows that there is X, Y and Z participants.

x : $X_c = sCRQ + pr * rCRP$

Participant x consumes the energy corresponding to the energy used by sending a CRQ plus the energy used on receiving a CRP from each participant replying.

y : $Y_c = rCRQ$

Participant y in is the energy consumed by receiving a concrete search request but not answering with a concrete reply.

Z : $Z_c = rCRQ + sCRP$ Participant z consumes the energy corresponding to the energy used by receiving a FRQ plus the energy used on sending a FRP.

E : $E_c = I * (X_c + pr * Z_c + (p - 1 - pr) * Y_c)$

The total energy consumed by a full search with reactive communication is equal to each inquiry multiplied with the energy consumed by sending the inquiry (X), plus the energy consumed by receiving the request packet and sending the reply packet (Z) for each participant that replies (pr), plus the energy consumed by receiving the request packet (Y) for each participant (p) minus yourself minus the participants who reply(pr).

Proactive communication energy calculation

For the communications with the proactive communication is shown on figure 8.4 where each square represents a mobile phone and each arrow between the mobile phones represents one communication.

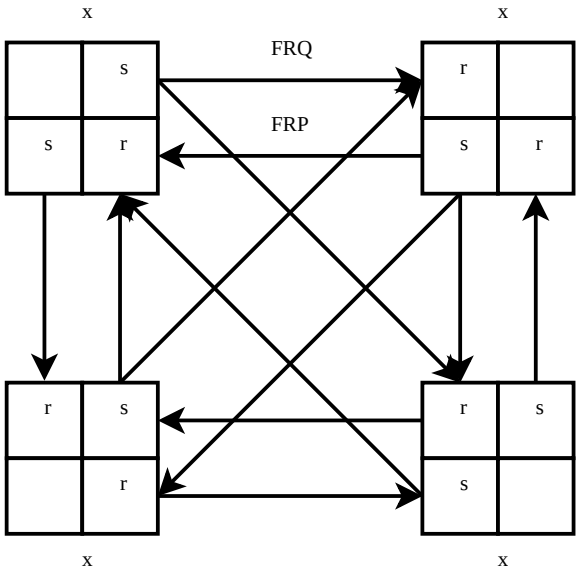

Figure 8.4: Proactive communication

With the proactive communication the energy consumed is the same for each participants so the energy consumed is equal the number of participants multiplied with the energy consumed per participant.

e : $e = H * (sFRQ + (p-1) * rFRP)$

The energy consumed per participant is the energy consumed by sending a full request plus the energy consumed by receiving a reply from each participant.

E : $E = e * p$

The total energy is the energy consumed per participant (e) multiplied by the numbers of participants (p).

8.3.1 Measurements

To be able to replace some of the variables from figure 8.2 with how much energy they consume, measured on a mobile phone, there is performed a test which includes sending a full request (sFRQ), receiving a full request (rFRQ), sending a full reply (sFRP), receiving a full reply (rFRP), sending a concrete request (sCRQ), receiving a concrete request (rCRQ), sending a concrete reply (sCRP) and receiving a concrete reply (rCRP).

There is performed four tests with two mobile phones, as shown figure 8.5 where mobile phone A sends 100 GET S request packets to mobile phone B. Underneath mobile phone A and B on figure 8.5 there is written which variables are measured on each mobile phone per test.

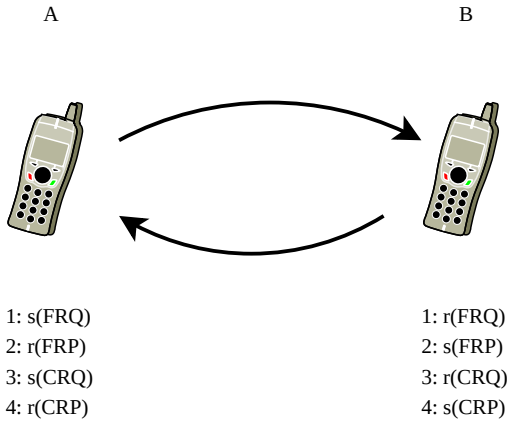

Figure 8.5: Energy measurement setup

The output from the tests is the energy consumed by sending or receiving a specific packet, which then replaces variables in the energy formulas used to calculate the energy consumed in a given network. In table 8.4 the measurements regarding energy consumed in mAh are measured with a program called Energy Profiler [16] developed by Nokia.

	A	B	A	B	A	B	A	B
1.Packet	sFRQ	rFRQ	sFRP	rFRP	sCRQ	rCRQ	sCRP	rCRP
2.Idle(mAh)	320	320	320	320	320	320	320	320
3.Running(mAh)	396	384	387	394	394	386	390	381
4.Task(mAh)	76	64	67	74	74	66	70	61
5.Time per Task(sec)	2	6	2	4	2	4	2	3
6.Packets s/r	100	91	100	75	100	98	100	95
7.Total s/r time(sec)	172	516	168	273	181	416	191	326
8.Packets per hour	2093	635	2143	989	1989	848	1885	1049
9.Energy per packet(mAh)	0,0363	0,1008	0,0313	0,0748	0,0372	0,0778	0,0371	0,0581

Table 8.4: Energy measurements table.

In table 8.4 the idle and running measurements are given as average values measured with Energy Profiler, the rest of the variables are calculated. In table 8.4 the amount of received packets are not the same as the amount send, presumably this has to do with the hardware buffer getting full because the platform itself if not able to process the data before the buffer gets filled. The data from Energy Profiler contains measurements from when the platform is started which means that there are data which are not relevant for the energy measurement, therefore it is decided to find a steady state where the mobile phone only uses energy on sending or receiving packets. This steady state area within the measurements are the basis of the average values.

- Packet is a list of all packets.
- Idle it the energy measured in mAh when the platform is started but not performing any tasks.
- Running when the platform is started and performing one specific task continuously, this is measured in mAh.

- Task is the energy consumed by a specific task measured in mAh.
- Time per task is the time it takes to perform a specific task.
- Packets s/r is the number of send or received packets.
- The total times is measured in how long it takes to send or receive 100 packets.
- Packets per hour is how many packets it is possible to send or receive per hour.
- Energy per packet is the energy consumed in sending or receiving one packet measured in mAh.

	Average(mAh)	Standard deviation(mAh)	Percentage deviation
idle	320	17,89	5,58%
sFRQ	396	20,71	5,22%
rFRQ	384	12,30	3,20%
sFRP	387	19,66	5,08%
rFRP	394	20,52	5,20%
sCRQ	394	21,75	5,52%
rCRQ	386	15,35	3,97%
sCRP	390	19,61	5,02%
rCRP	381	17,60	4,61%

Table 8.5: Standard deviation for idle and running values from table 8.4.

Table 8.5 shows the standard deviations for the idle and the different packets.

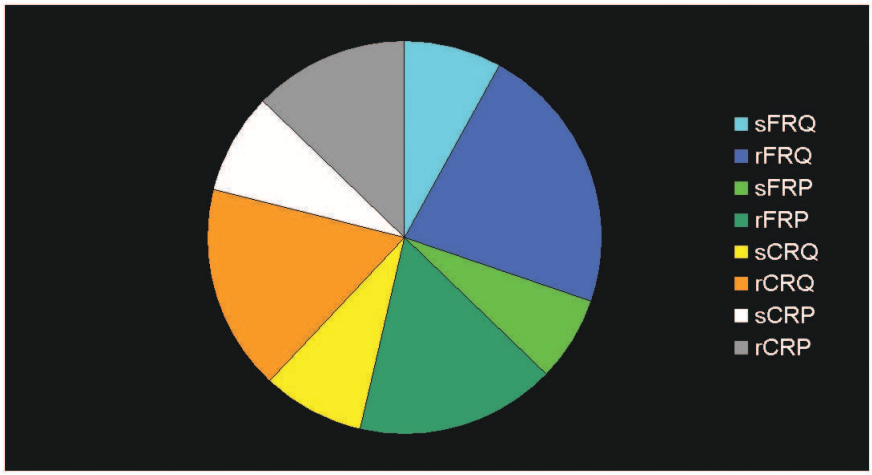

Figure 8.6: Energy per packet comparison.

Figure 8.6 shows each of the different packets compared to each other. From the figure it is possible get a quick overview of that the rFRQ packet consumes the most energy measured in mAh, and the sFRP packet is the packet consuming the least amount of energy measured in mAh.

8.4 Energy consumption parameters

In this section the energy consumption parameters are defined based on packet and communication types. The energy consumption parameters are the variables which can not be measured or calculated from Energy Profiler, but the variables which are depending on the users. This is how many inquires are send and how many participants the network consist of.

Reactive	Common	Proactive
Number of inquiries	Participants	Advertise frequency
Number of FRP's	Time period	
Number of CRP's	Packet loss	

Table 8.6: Energy consumption parameters

Table 8.6 shows which parameters apply for the reactive communication, which are common for both communication types and for the proactive communication.

8.4.1 Reactive parameters

In the reactive communication type the parameter which leads to the highest energy consumption are the number of inquiries send. Each inquiry send will effect all participants in a network. When an inquiry is send there are two options to chose from. The first option is to send a GET S request packet for all services, which means that all the participants receive this packet and replies. The second option when making an inquiry is to send a GET S request packet for a specific service. All participants receive the GET S request packet but only the participants with the service makes a reply.

8.4.2 Common parameters

The common parameters are the number of participants, the time period the network is active, which is from the first time a packet is send to the last packet is received and the packet loss. The packet loss is given in the percentage of how likely it is for a packet to reach its destination.

8.4.3 Proactive parameters

In the proactive the only parameter which is possible to adjust is the advertise frequency, meaning the period between the packets which are send.

8.5 Scenario specifications

Before the energy consumption in the two scenarios is calculated, some basic informations are defined. The packet loss percentage is transformed into the probability that a packet is send and received successfully. This is used to calculate the number of retransmissions needed to guarantee a specific success rate, of each participant having received the inquiry, and the one who send the inquiry has received a reply from each participant.

Another specification which is needed, is the time from an inquiry is send to the time the reply is received. This will determine for how long the user will have to wait in worst case.

8.5.1 Packet loss probability calculations

The probability is calculated on the basis of an assumption, that a given network has a given percentage of packet loss due to the traffic on the network. So the probability calculations are made on the assumptions, that the network has a packet loss percentage of 10%, 50% and 90%, for both the a reactive and proactive communication types. For example if there is 90% packet loss on a network, it could be because every body is using the network for download. This could result in packets not reaching their destination.

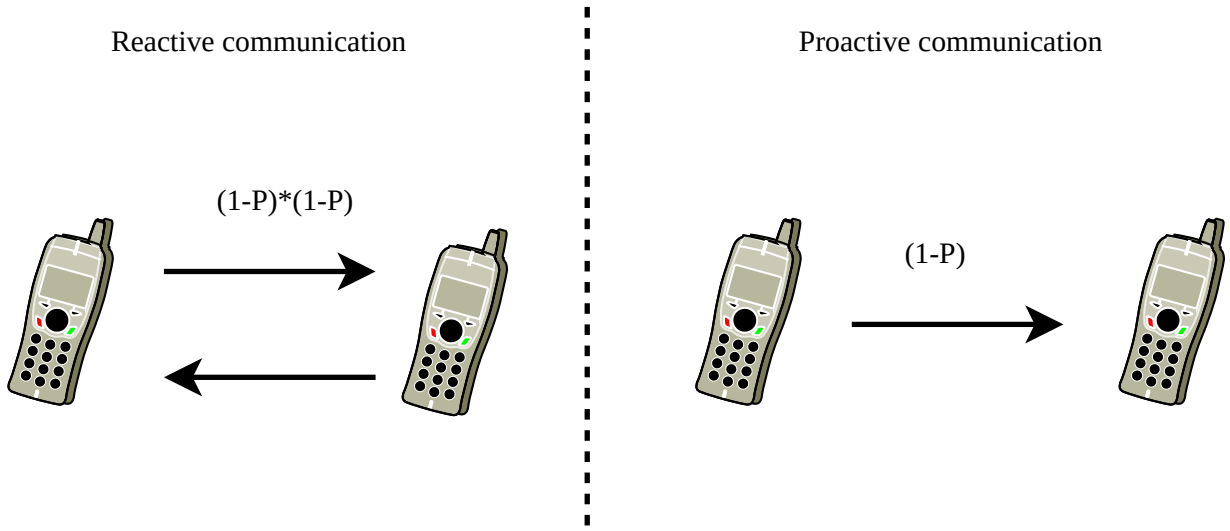

Figure 8.7: Probabilities for reactive and proactive communication.

Figure 8.7 shows the probabilities for the two communication types. Here p is the probability of success, so for example if there is a packet loss of 10% with proactive communication the probability of a packet is received is 1 - 0.1 = 90%, which means there is a 90% guarantee that the packet is send and received successfully in first transmission.

	Reactive
1. transmission	$Pr(n=1) = (1-p) * (1-p)$
n transmission	$Pr(n=N) = Pr_{N-1} + ((1-Pr_{N-1}) * Pr_{n=1})$
	Proactive
1. transmission	$Pr(n=1) = (1-p)$
n transmission	$Pr(n=N) = Pr_{n=1} + ((Pr_{n=1})^n * (1-Pr_{n=1}))$

Table 8.7: Probability formulas for both networks.

The formulas from table 8.7 are plottet in a graph in figure 8.8.

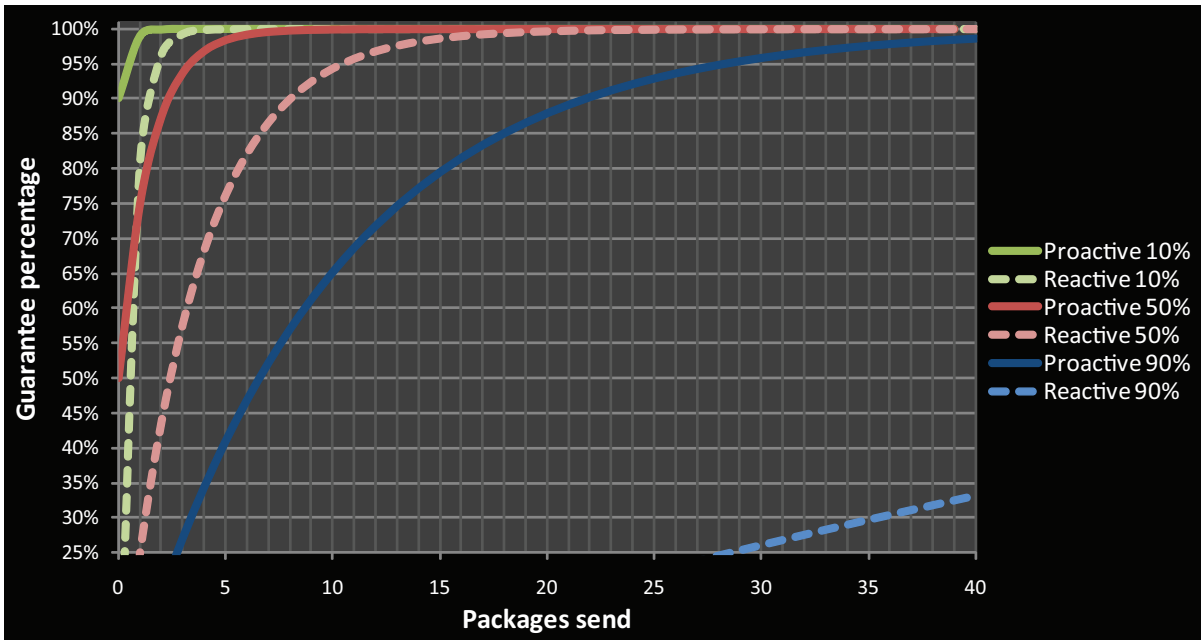

Figure 8.8: Probability calculations.

In figure 8.8 the horizontal axis is the number of transmissions, and the vertical axis is the success rate. There are made a curve for each communication type at a given packet loss percentage. It is now possible to read from the graph in figure 8.8, that the reactive communication with 10% packet loss after three transmissions, have 99% guarantee that all participants in the network has received the inquiry and replied it. The graph shows for each inquiry send the probability that it reaches its destination moves towards 100%.

8.5.2 Send receive time

The time from when an inquiry is send to it is received, helps to determine a worst case scenario for the time a user must wait, based on the number of participants replying. It also determines the time period which the proactive communication can use to set the H (frequency between transmissions). The time measurements are based on observations made during the measurements in section 8.3.1 and an additional test. The additional test was to determine the time is takes to send an inquiry and receive the reply. The observations made during the measurements in 8.3.1 where the time it takes to process a message which is already in the hardware buffer.

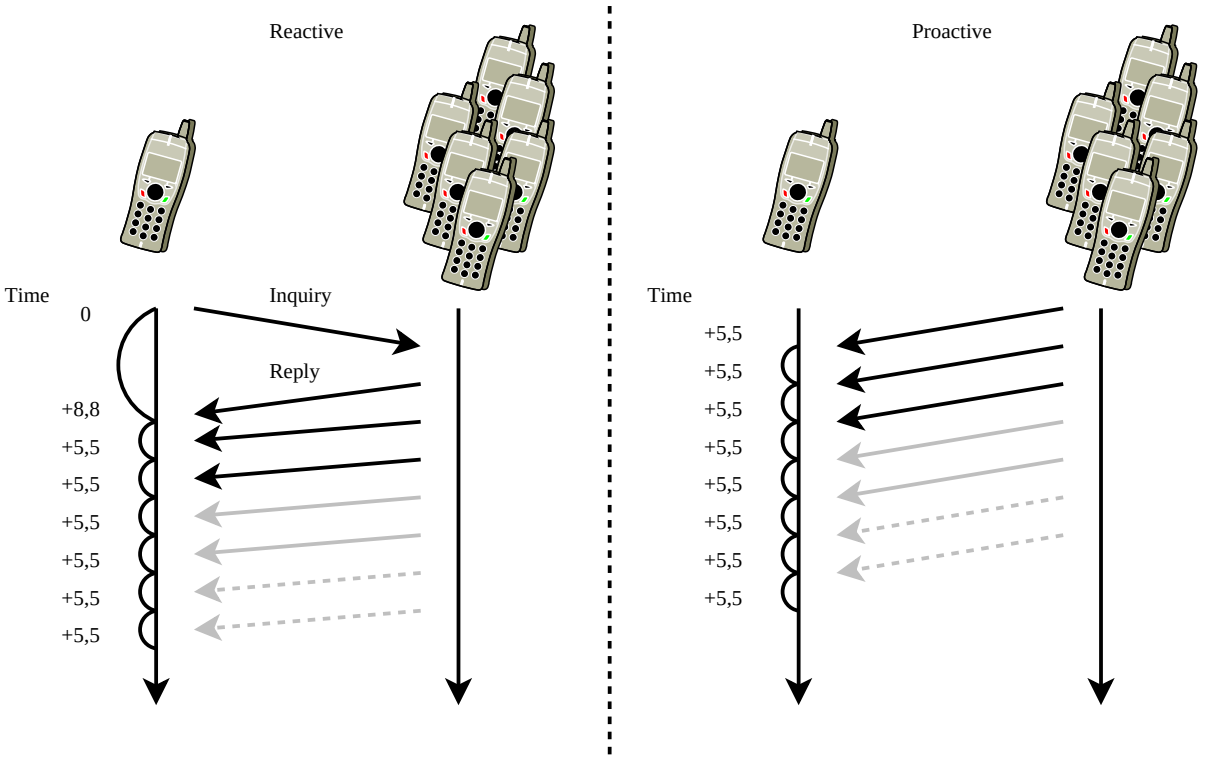

Figure 8.9: Send receive time.

The additional test shows that the time from the inquiry is send to the reply is received in a reactive communication, takes 8,8 seconds and the following replies received as shown on figure 8.9 takes 5,5 seconds per reply to process. If they are to be received exactly after each other. For the proactive communication it takes 5,5 seconds for each inquiry received. These time calculations are for a minimum of two participants (p).

$$ReacitveTime = 1 * 8,8 + ((p-2) * 5,5)$$
$$ProacitveTime = (p-1) * 5,5$$

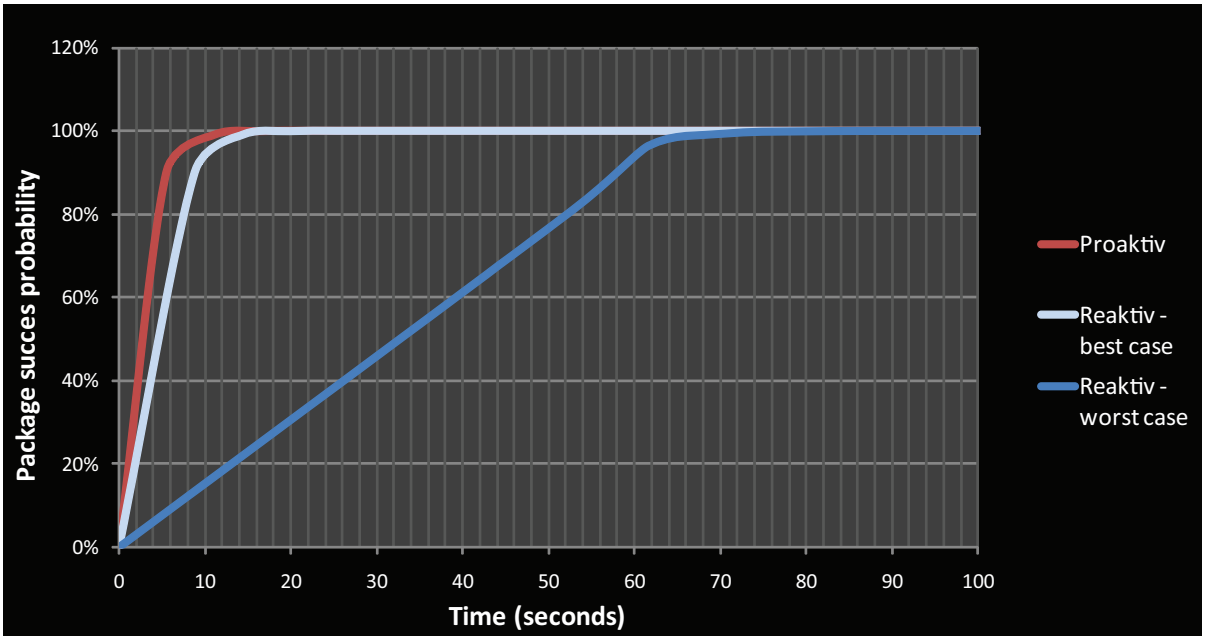

Figure 8.10: Guarantee mapped into time.

Figure 8.10 illustrates the time it takes in a network with ten participants to guarantee that 100% of all participants with reactive communication has received and replied an inquiry. For the proactive communication the figure shows how long it will take for one participant to know all the services on a network. For both the reactive and proactive communication the processing time only includes processing data once, so if the same data is received more then once it is not taken into consideration. Also for the proactive communication it is expected that all participants are able to send the amount of packets it takes to be 100% sure that all has received the packet, within the time it takes to process one message from each participant. Figure 8.10 is based on the probabilities from figure 8.8 and the time calculations from 8.5.2. With a reactive communication and 10% packet loss it takes 79,2 seconds to guarantee that 100% of all participants has received and replied an inquiry. For the proactive communication it takes 49,5 seconds to guarantee that 100% of all participants has received an inquiry.

8.6 Low density network

This scenario is to illustrate the energy consumed in a low density network. This network only has ten participants. In a low density network the developers presume that the user most likely will make inquiries asking for all available services with the reactive communication. In the scenario the reactive communication will be compared to the proactive communication in order to determine when it is more energy efficient compared to energy consumption, to use one communication instead of the other.

In this scenario it is decided by the developers that there must be a guarantee above 90% that all participants has received the inquiry and replied. Based on the graph from section 8.5.1 with reactive communication to guarantee above 90% of all the participants received and replied the inquiry there must be send at least two inquiries. With reactive communication and a packet loss percentage of 10% by sending two inquiries there is a guarantee that 96% of the participants has received the inquiry and

replied it. When sending two inquiries with proactive communication and a packet loss percentage of 10% there is a guarantee that 99% of the participants has received the inquiry and replied it.

Best case - The best case is the energy used one inquiry.

Worst case - The worst case is the energy used be sending the number of inquiries which can guarantee that above 90% of all participants has received and replied the inquiry.

Energy consumed per period

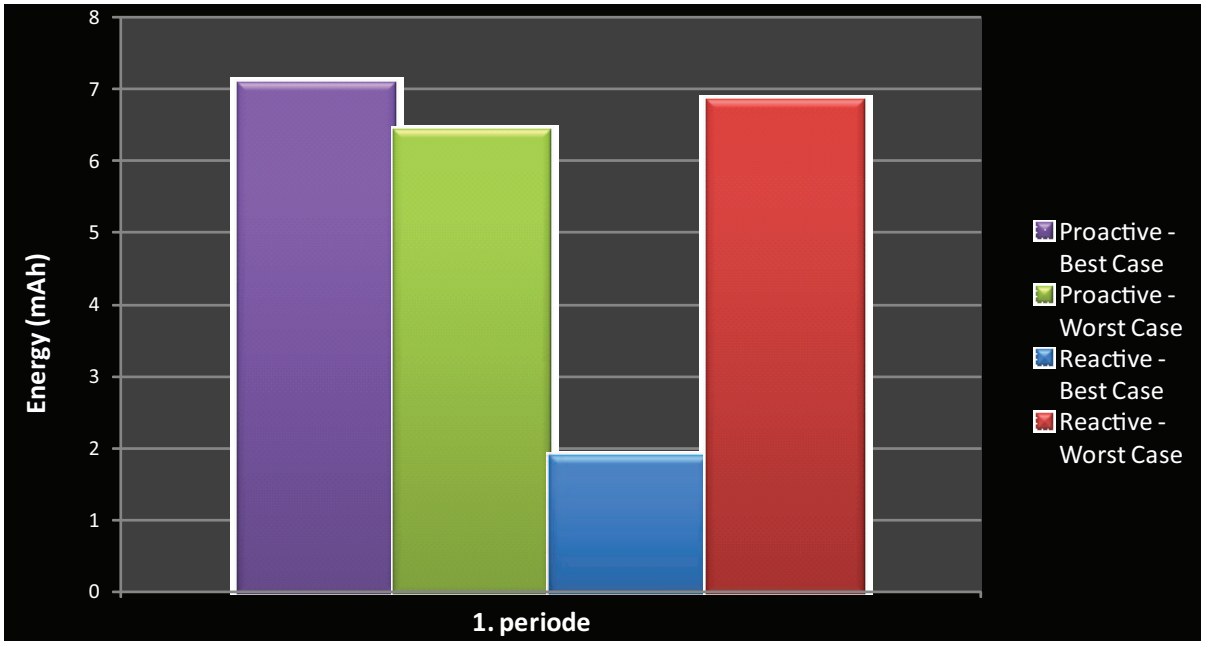

Figure 8.11: One time period for low density.

Based on the numbers of participants and the time calculation from 8.5.2 the energy consumed over one period time. The period time with reactive communication is the time from the first inquiry is send to the last reply is received. The period time with proactive communication is the time between inquires. Figure 8.11 shows how much energy is consumed during one period for each of the communication types best case and worst case.

Energy consumed over 60 minutes

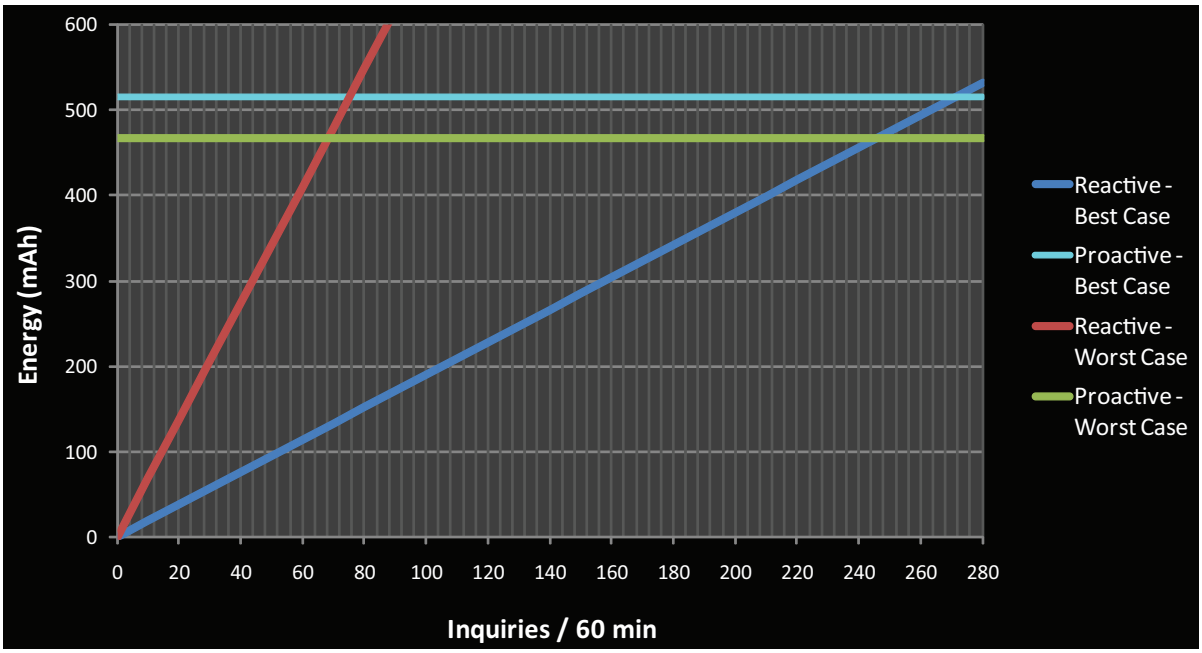

Figure 8.12: Energy calculations for a low density network.

Figure 8.12 shows the energy consumed in best case and worst case for a reactive communication and a proactive communication. In figure 8.12 the horizontal axis are number of inquiries per hour and the vertical axis is energy consumed in mAh. Figure 8.12 shows how many inquiries with reactive communication there is needed to consume more energy then with proactive communication.

The frequency for the proactive communication is set to the period time it takes to receive an inquiry from all participants minus 1. The proactive communication uses the same amount of energy at any given time because it sends inquiries with the same frequency continuously. The reactive best case shows the energy used if all participants received an inquiry and replies it the first time. The reactive worst case shows the energy consumed if the reactive communication has to be above 90% sure that all participants has received and replied the inquiry.

Energy consumed to ensure above 90% guarantee

To determine which frequency the proactive communication should have between sending advertisements, the energy over time for the entire network is mapped on figure 8.13.

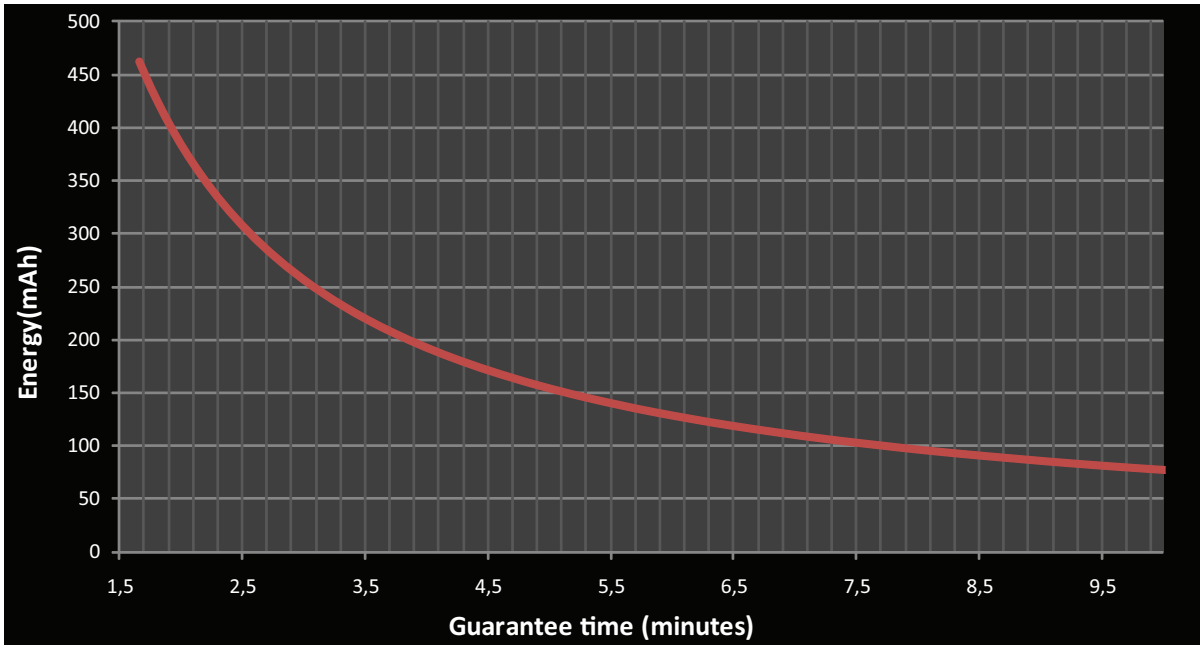

Figure 8.13: 99% guarantee proactive communication in the network mapped into energy over time.

On figure 8.13 the energy consumed over time is mapped for 99% guarantee that everybody on the network knows all available services to a certain time. From this figure a trade off between energy consumed in the entire network and the time before there is 99% guarantee that all participants know all available services must be made. This is in order to determine at which frequency the proactive communication must advertise. The developers has decided that 7 minutes and 30 seconds between each time all participants are updated with all available services on the network. At 99% guarantee time over 7 minutes and 30 seconds the entire network consumes 102,77mAh. Because all mobile phones in a network are not synchronized to advertise at the same time, the 7 minutes and 30 seconds is the maximum time it can take to know 99% of all available services. It is possible to receive advertisements from other mobile phones within these 7 minutes and 30 seconds.

Energy consumed in a low density network

The energy consumption calculations this scenario are based on the energy formulas from section 8.3 together with the measurements from 8.3.1, the probabilities from section 8.5.1 and the time calculations from section 8.5.2. Furthermore the period time for the proactive communication is set to 7 minutes and 30 seconds. The network is look at over a time of 60 minutes.

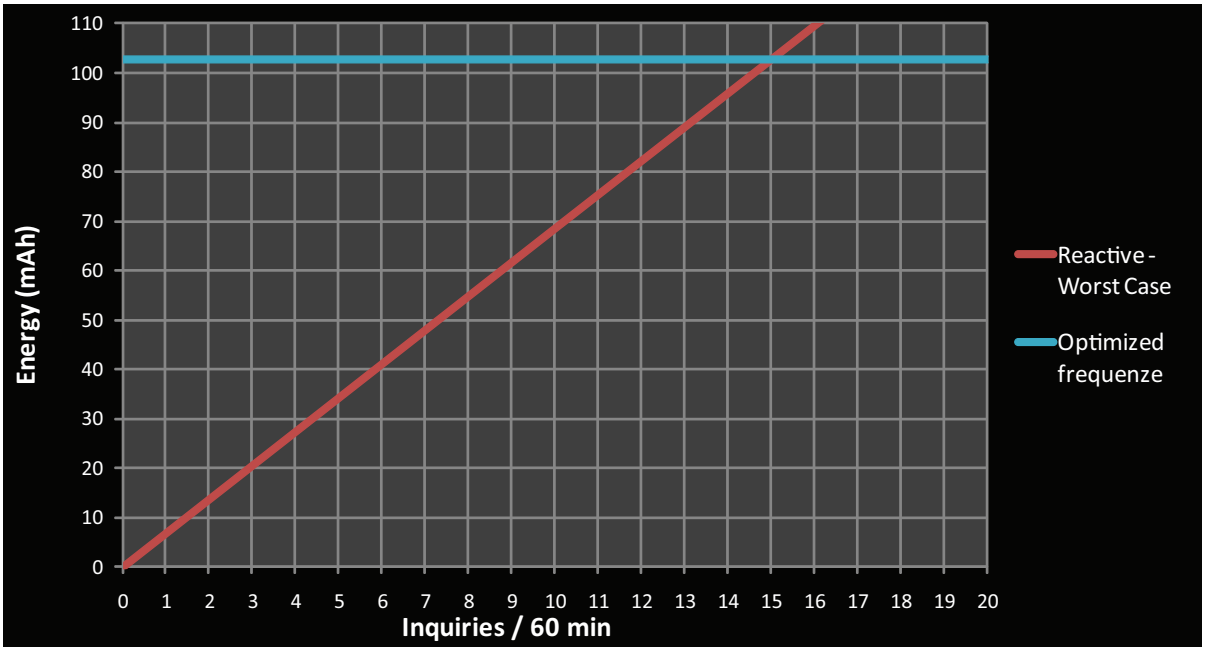

Figure 8.14: The networks over time (60 minutes).

Figure 8.14 shows the network over a time period of 60 minuets for reactive and proactive communication with a guarantee above 90% of all participants has received an inquiry and replied(reactive) or above 90% of all participants (proactive) knows all available services on the network. The graph shows that in low density network with ten participants, where the proactive communication has a frequency of 7 minutes and 30 seconds it is more energy efficient to have the proactive communication if there is more than 14 inquiries with reactive communication on a network within 60 minutes. If the time between advertisements with the proactive communication was shorter the energy consumed would be higher and the energy consumed would be lower if the time between advertisements was longer. If there where more than ten participants the energy consumed in the network for both communication types would be higher.

8.7 High density network

This scenario is to illustrate the energy consumed in a high density network. This network has 50 participants. In a high density network the developers presume that the user must likely will make an inquiry asking for a specific service with the reactive communication. The approach to determine the energy consumed and when it is more energy efficient to used one communication type over another is the same as in scenario one 8.6. Due to that in the network the user sends inquiries for specific services the calculations are based on 10% or 25% or 50% of the participants in the network who are able to reply the inquiry because they have the specific service.

In this scenario due to the density of the network the developers has decided to calculate the energy consumed with a packet loss percentage of 50%. With the reactive communication to guarantee above 90% of all the participants has received and replied the inquiry must be send at least nine inquiries. For the reactive communication with a packet loss percentage of 50% by sending nine inquiries there is a guarantee that 92% of the participants has received the inquiry and replied it. When sending four inquiries with the proactive communication with a packet loss percentage of 50% there is a guarantee that 94% of the participants has received the inquiry and replied it.

Energy consumed per period

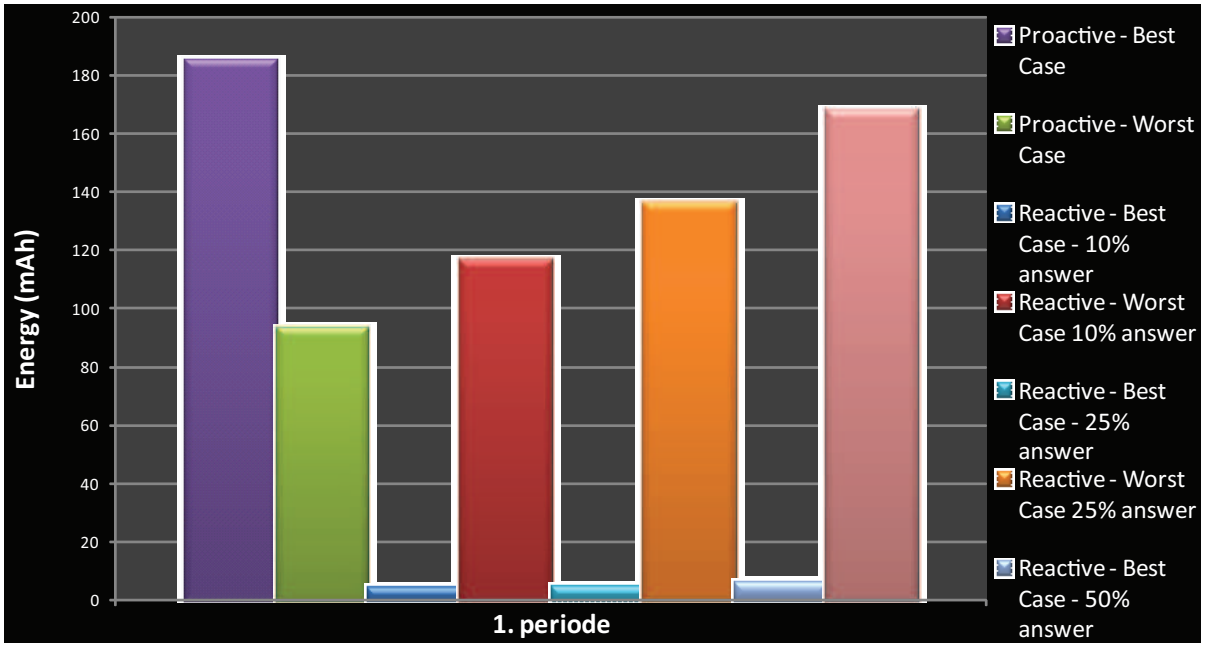

Figure 8.15: One period time for high density.

Energy consumed over 60 minutes

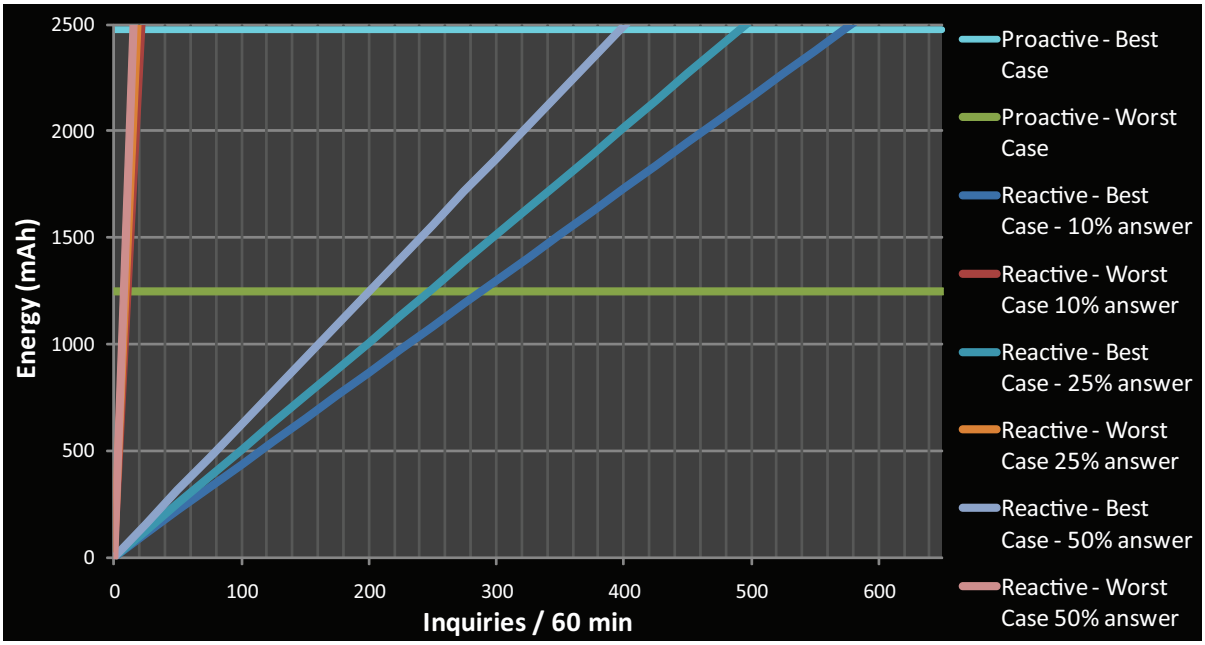

Figure 8.16: Energy calculations for a high density network.

Energy consumed to ensure above 90% guarantee

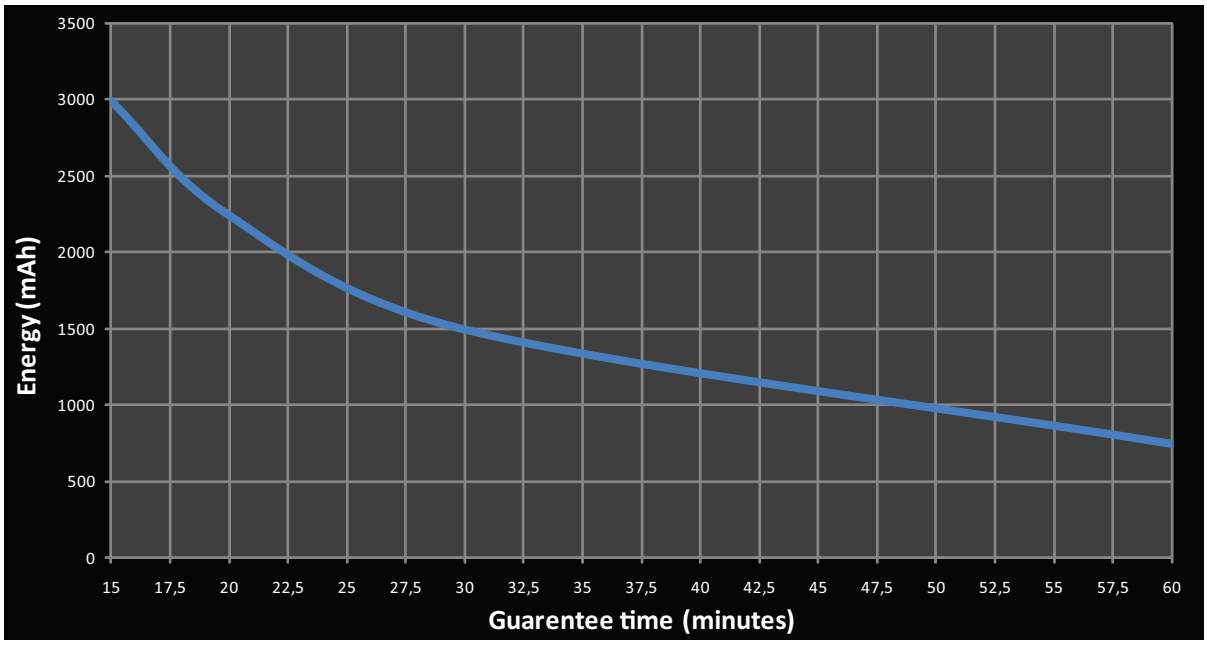

Figure 8.17: 94% guarantee with proactive communication in the network mapped into energy over time.

The developers has decided the proactive communication shall only advertise every 30 minutes the energy consumption considered. The 30 minutes is like the low density scenario based on a trade off between energy consumed in the entire network and the time it takes to know above 90% of all available services.

Energy consumed in scenario two

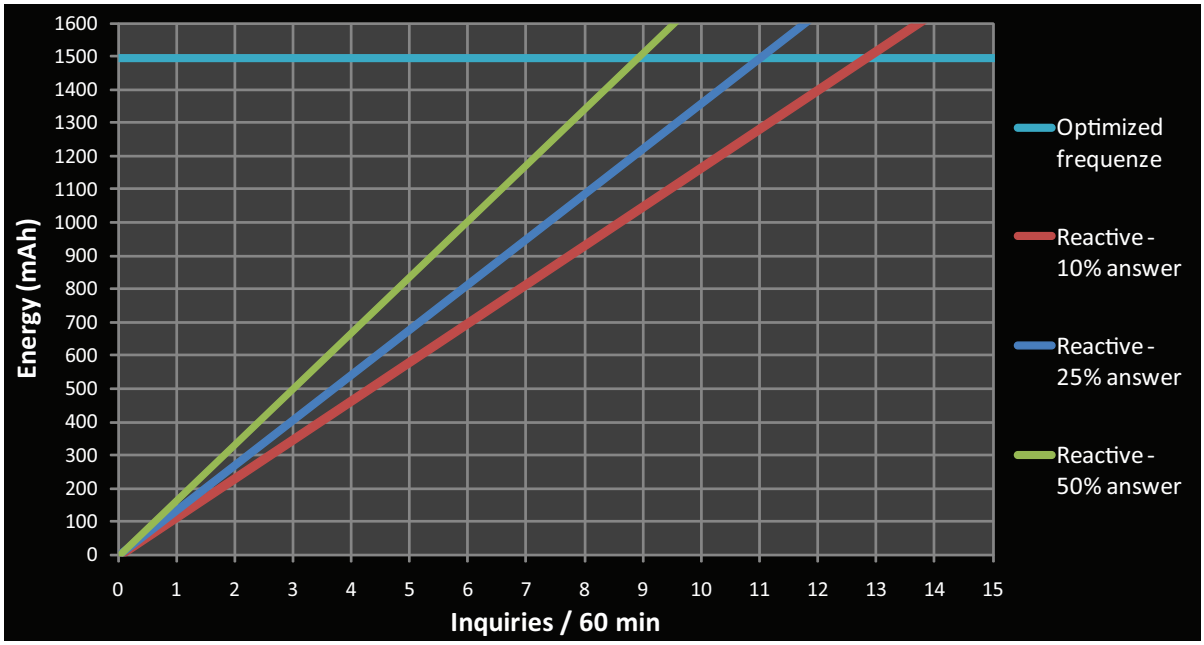

Figure 8.18: The networks over time (60 minutes).

Figure 8.18 shows the high density network over a time period of 60 minutes with both reactive and proactive communication types. The reactive communication is showed for 10%, 25% and 50% of the participants which answers an inquiry. With 10% of the participants which answers an inquiry it is possible to send 13 inquiries before the reactive communication would consumes more energy then the proactive communication. With 25% answers it is possible to send 11 inquiries, and for 50% answers with reactive communication it is possible to send nine inquiries before it consumes more energy then the proactive communication.

8.8 Network communication type conclusion

There are two issues with the reactive and proactive communication considered in the scenarios. The first issue the time it takes from joining the network to all available services are known, this is referred to as the guarantee time. The second issue is how much energy the mobile phone consumes.

The reactive communication has the advantage of using a small amount of energy on the network if only a few users on the network wants to know what services are available over a longer period of time. The disadvantage is that all users must ask for services, so if a lot of users always ask for available services, the reactive type consumes a large amount of energy by all users when requesting and replying. This is the complete opposite with proactive communication, where if all users always wants to know all services, the proactive type does not need to make a request to tell other users what they have, this type advertises services anyway and thereby no request is needed and the amount of used energy is minimized. The disadvantage is if only a few wants to know what services available over a longer period of time, this type uses a large usage of energy by advertising information that is unnecessary.

The energy used in best case with proactive communication, whether it is in a high or low density network, is for one advertisement always more expensive as one request and reply in both best and worst case in with reactive communication. This is shown in figure 8.11 and 8.15. This is because all users in with proactive communication in best case always sends and receives an advertisement for every frequency. The reason for the proactive worst case energy usage always is lower then reactive worst case, is that when worst case proactive advertisement is made, everybody will send one but not everybody will receive one, depending on the packet loss. The reactive sends more then one packet in worst case, to ensure success for the packet received and replied.

At the button line the energy consumed in the network is, if a lot of users wants to know the services often the proactive communication is the best approach, and if a few users over a longer time wants to know the services reactive communication is the best way. This exact amount of request over time, from when it is more desirable to chose one type instead of the other, is express in the scenarios in the energy calculations figures 8.12 and 8.16. The low density shows that in worst case close to 70 full request for all services in with reactive communication, before it is more desirable to make a proactive type instead. The 60 requests corresponds to the 10 mobile phones can ask for services almost 7 times in an hour. In the best case the users can ask more then 27 times over an hour. The high density network shows that in worst case less then 10 concrete request for a specific service in a reactive type, regardless how many that answers, before it is more desirable to make a proactive type. The 10 requests corresponds only to 5 of the 50 mobile phones can ask for an hour. In the best case the users can around once each, depending on how many answers, over an hour.

The guarantee time is measured different when looking at the network communication types. In reactive communication it is not a problem to ensure when a user is able to know all available services, because it is only a matter of sending a certain amount of requests regardless network size. An example of this is made in figure 8.10, where it shows the time delay for a low density network with no packet loss to 10% package loss. In proactive communication the guarantee time is considered that only one advertisement is sent for each frequency, this is also the way it is done in the scenarios. The other way was to raise the amount of advertisement per frequency, which use a lot more resources per frequency, but then the frequency could be lowered compared to the ones in the scenarios and till meet the same guarantee. The disadvantage of this is that the information needed could already be received in the first frequency, and thereby saving energy over time. The time for the requester to process replies is an issue

for both communication types, but this is an issue which is not considered in this analysis.

The guarantee time for the proactive communication type is ensured by waiting for a certain amount of frequencies where one advertisement is sent per frequency. Instead the number of advertisements needed to ensure that all participants receive the advertisement could be send every frequency. This would give a faster time for the guarantee, but increase the amount of advertisement and thereby increase the energy consumed per frequency. The other advantage which is stated in this report, is that the advertisements which are received randomly and therefore there is a chance that the service which the user wants to use is received as one of the first advertisements, if this was the case the following advertisements would be unnecessary energy consumption. This principle could also be used for the reactive system, thereby meaning that the requester could make another request, if the wanted service is not among the once received in the first request, instead of making multiple requests for ensuring all services available are shown. This would then save a lot of energy by not sending multiple requests.

The guarantee time for the proactive communication can be optimized in energy usage versus guarantee time, if the size of the network is known. This is set up in figure 8.13 and 8.17, where also the packet loss is included, where the proactive communication includes the extra needed advertisement, for ensuring packet loss, per frequency. The longer the user is able to wait, the less energy would be used over time. It clearly shows that the bigger network a is, the longer time the user have to wait for ensuring all available services are shown. Though, as earlier mentioned, could the needed service already be shown in the first advertisement received.

The frequencies decided from how long a guarantee time versus energy usage in the proactive communication, is set into the worst case for the reactive communication in figure 8.14 and 8.18. They show that if the user is able to wait the decided guarantee time, even fewer reactive communications request can be made. This gives that over time, the proactive communication type is more desirable than the reactive communication type, especially in high density network where, regardless how many replies there are, no more then 14 request can be made before the proactive frequency is more desirable compared to energy consumption.

CHAPTER 9

Conclusion

Through out this report there is analysed, designed, implemented and tested a service discovery platform which answers the question asked in the initial problem.

"Is it possible to develop an universal application for a mobile phone which can discover and use other mobile phones services and features?"

This conclusion will address how the initial problem is fulfilled, further more it will address issues which are not documented in the report. These issues are issues which the developers have encountered during the development of the product with the programming language pys60. After this the platform analysis is concluded upon.

In the introduction new phenomenons as piggybacking, app stores and the green wave are introduced. From the introduction there can be read that 98% of all danish families has a mobile phone. The introduction also shows that the mobile phones is not only used for calling and texting any more, but they are also used to take / display photos, record small movies, surf the internet, GPS and so on. All these new features creates new trends and new markets for applications developers for mobile phones. The mobile phone manufacturers each have their own app store where it is possible to download applications for their mobile phones. This report shows how to make a platform which can work on all mobile phones and share service between them, so a mobile phone which for example does not have GPS could ask another one with this service to share its GPS coordinates.

Based on the introduction and the initial problem statement, requirements to a platform which can fulfil the initial problem are made. These requirements are what the developers found necessary for making such a platform. There are requirements to what kind of network the platform must use to communicate between mobile phones. There are also requirements to the platform itself, of how it discovers and shares services, what kind of informations there must be available in order to communicate between mobile phones. If a service is shared it must also be able to distribute a client to the service. Before the developers designs the platform they looked at existing protocols for service discovery, this was to find functionalities from the existing protocols which the developers could use for the platform. The most interesting existing protocol which has contributed most to the platform is UPnP. From UPnP the developers chose to use some of the same principles for the discovery and advertise processes.

From the requirements and the information gathered from the existing protocol analysis there is made a design. The design covers the platform and is illustrated with deployment diagrams, use cases and activity diagrams. The design consists of two main parts of the platform. The first part is the Discover Module which handles everything regarding when a user wants to search for any available services on a network. The Discover Module is designed to assist the user in getting a list of all or a specific service on a network and if the user does not have the service installed the Discover Module also retrieves a client to use the service. The second part of the platform is the Share Module. The Share Module is designed to

share the services which a user decides to share. So if a user asks for available services on a network the Share Module makes a reply with services it is sharing. There is also made a design draft for a service in appendix A, this is a service designed to be shared on the platform. The services is called W3S and is for sharing a 3G data connection via WiFi to users of the service.

The implementation is made on the basis of the design. The design is used to make the classes and functions needed in order to get the implementation of the platform to fulfil the requirements for the platform. The implementation chapter only describes the additional things which are necessary to get the implementation to communicate between functions in classes, also what data are send back and fourth between the Discover Module and Share Module. Besides the additional implementations the platform is implemented as designed. To make sure the implementation of the platform works as designed the accept test is performed. The accept test is based on the requirements for the platform. The platform implementation passes all tests in the accept test.

Through out the implementation there has been a lot of complications concerning the pys60 programming language. Recently the people behind pys60 has made a new version of pys60. The new version is version 1.9.7 which includes the 2.5 core of Python. The pys60 language is based on C++ for symbian s60 which means that there is a lot of libraries which are preimplemented meaning they are not easily modified. Pys60 is strong for quick prototyping because of all these libraries, but in the new pys60 the implementation concerning thread programming was so full of bugs that is was not possible to implement the network thread with a nonblocking parameter. The nonblocking parameter makes it possible to check if there are any data in the WiFi hardware buffer, if there is any data the thread retrieves it, if not it sleeps and checks again later. This nonblocking functionality works in an older version of pys60. The older version is 1.4.5 which in an implementation of Python version 2.2.2. Pys60 version 1.4.5 does not support any XML libraries which version 1.9.7 does. So in order to get the network thread to work properly it was decided that is was more important to use version 1.4.5 and then make our own XML parser specific for this project.

The platform analysis makes a thorough analysis of the platforms service discovery processes. The service discovery communication type the platform implements, is reactive communication, and the platform analysis compares this to how a proactive communication would consume energy and time from joining the network to knowing all available services, the guarantee time. The comparison is done for a low and high density network, where the frequency with proactive communication is optimized to lower energy consumption versus longer guarantee time. The parameters available when a trade off between energy consumption and guarantee time is made, are for the reactive communication the numbers of inquiries send and the frequency for the proactive communication. Common parameters are also the amount of participants, the time a network is active and the packet loss. The packet loss is a parameter considered for itself, and shows that roughly twice as many request must be made in reactive communication compared to advertisements in proactive communication, to ensure the same guarantee percentage.

The reactive communication shows to be good for a low density network, where few wants to know all services over time. The proactive communication shows to be good in a high density network, where all participants wants to know all available services always. Between these two outer points of network density and users wanting to know available services, there is a grey zone. In this grey area participants who wants to know a specific service and only few of the other participants have it, then it can be more desirable in terms of energy consumption to make a reactive communication in high density network. Though the more users that have the service, the amount of replies for the concrete request increases. This will at the end end up consuming the same amount of energy as a request for all services. Looking

over a long period of time, proactive communication is never desirable in energy consumption for low density networks, because of the continuous advertisement of services.

To ensure packet delivery in a network is very difficult, because the packet loss percentage can differ much. Often when it is a big network, the packet loss is high, but it can also be high in low density network when a user uses the network by for example downloading, or if a there is bad signal because of other networks, building etc. Some sort of packet loss percentage measuring could be integrated, so the network self finds out the packet loss and thereby adjust the amount of send packages to ensure guarantee of delivery. Such measuring is difficult to make with a UDP network, because there is no guarantee of package delivery like in TCP. On the other side if the wanted service is not shown on the list, a new search is always an option.

A problem with the platforms guarantee time, which is not considered in this report, is the time it takes to handle the packets. If this handle time could be decreased, then the guarantee time could be lower. By the experience made during tests and implementation, it is most likely that the databases used between the processing threads that gives the problem. If the process time could be decreased from the now 5,5 seconds to less then 1 second, by using a faster buffer, this problem could maybe be eliminated. Another problem is the energy consumed when the platform is in idle mode. It measured during tests to consume over 300 mAh, which is a lot compared to the processes a packet must through consumes less then 0,1 mAh. The battery for the Nokia N95 8GB is only 1200 mAh, which with the platform running, only gives about 4 hours of battery time with the platform running in idle mode. By the experience made during tests and implementation together with experience, it is most likely something to do with the process using WiFi which attend to use a lot of energy.

9.1 Perspective

With a platform like the one described in this report the ideas for improvement and expanding are many. It would be obvious to start with the things which where delimited from and look at some of the things which are delimited to and by that make the platform more universal in terms of use. Because the platform is a prototype there is not implemented any security when the mobile phones communicate between each other or on the mobile phone with encrypting any of the data on the phone like passwords for services. If the platform should make it from prototype to a version which could be sold to customers the security issue would be something which needs to be implemented. If the platform should be able to be installed on other mobile phones besides the Nokia N95 8GB which was used for this project, the platform should be developed in another programming language then pys60, maybe in Java which is supported by all mobile phone brands. It would also be a good thing to expand the support of the scenarios from figure 2.6 from only supporting the first scenario to supporting all three scenarios.

In the future the platform needs to be optimized to consume less energy then it does now. This is not only on a network basis but also internal. All the processes which makes the platform able to work needs to be looked through, to ensure they use the minimal amount of energy possible, without making the user wait for hours to get a reply on a network. On the platform there could be implemented an algorithm, which could calculate when it is less energy consuming with a proactive communication than a reactive communication. This should be some sort of dynamic distributed system, where devices could tell each other wetter to make a proactive or reactive communication. The problems of making such a system would, like the already known issues with proactive and reactive communication, be to know how many participants the network consist of and how many who wants to know the services available. An

additional problem, when running the proactive communication, is to decide a frequency so the system consumes less energy, but still have a reasonable guarantee time. This is issues already dealt with, which means the data from this report could be used to develop an algorithm for such a system.

Another optimization for the proactive communication, could be to change the advertisement by sending with a frequency to instead sending an advertisement when entering network and when leaving the network send a goodbye message. This is no problem in a low density network, where properly every user is in the same cluster, but when a high density network is considered, there could be a lot of clusters on the same network. It is then a problem to advertise when entering one cluster and leaving another, because it is not possible to detect when they are all on the same network. The jumping from one cluster to another, must be considered to be done often in a high density network with users moving around. This problem could be solved by using a lease time for the advertisement, meaning an arriving message only lives a certain amount of time before a leave message or new arriving message must be send.

The platform is a software system that makes it very easy to distribute software, but also to develop software that can be used to communicate between mobile phones and use native hardware on the mobile. By making an application that can start as server or client, it is easily used, shared and distributed by the platform. This gives an application, there if it was to be distributed as pre-installed software on a mobile phone, share every possible application on a free of charged ad hoc network, that does not require anything except a basic WiFi module. The profit could then be maid for when a user wants to use another users service, or when a share of a service is made, or a completely different way. The bottom line is, that it also gives a great opportunity to make money of this concept.

APPENDIX **A**

Service design draft

A.1 The service

Here the use cases and their activity diagrams for the service is described. The basic informations for W3S which needs to be configured from the platform are:

- If the mobile telephone which has the service installed has a 3G net subscription which allows the user to share it with others.
- If the mobile telephone as WiFi capabilities.

The explanation of elements in the use cases diagrams and activity diagrams made for the platform in section 4.3, does also apply for this chapter.

A.1.1 The Use Service

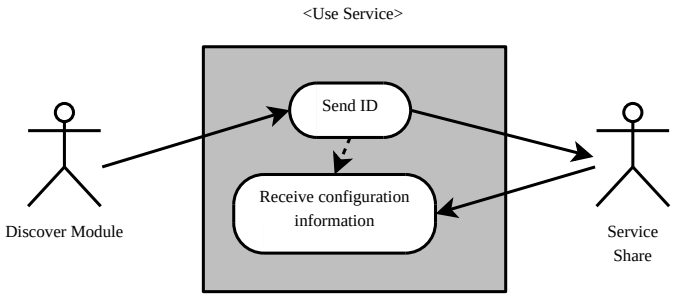

Figure A.1: The use service use case diagram

The Use service is when a user uses the W3G service. Figure A.1.1 shows the use cases which are associated with the Use Service node from the deployment diagram on figure 4.4. The use case diagram have two actors, the Discover Module and the Service Share. The Discover Module is the active actor which starts the communication and the Share Service is an passive actor which only responds on a requested communication.

Send ID

The *send ID* gets started by the Discover Module. This is done after the user has found a service which the user wants to use. Figure A.2 is for the service W3S, where first the ID and password is collected for

the W3S subscription. If the user have yet the enter the ID and password for the subscription an error message is shown, else the use service node connects to the service share node and sends the ID and password. The use service node then waits for a reply from the service share node which contains the configuration information to be able to use the W3S service.

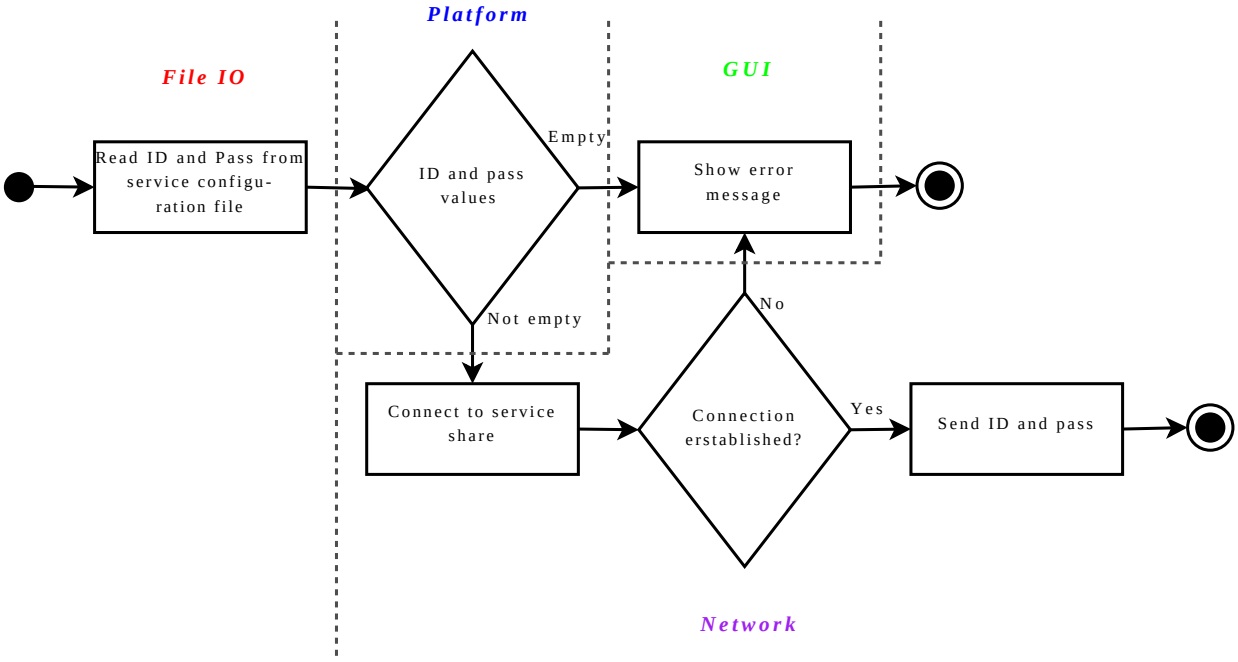

Figure A.2: The send id use case

Figure A.2 contains four different boundaries, File IO, Platform, GUI and Network. Each of the boundaries classifies which activities belongs to which boundaries.

File IO - File input output is for reading the file containing the ID and password.

Platform - The Platform makes sure that the service can retrieve the ID and password and if the value are empty notifies the user.

GUI - The GUI shows the error message for the user.

Network - The Network handels all communication over WiFi.

The activity diagram has two endpoints. The one of the endpoint is reached if an error has occurred during this activity, the other endpoint is reached if the activity is completed successfully.

Receive configuration information

The *Receive configuration information* is started if the send ID activity is completed with success. As shown on figure A.3 first a timer is started, then the timer is checked is it have not counted to 5 seconds yet. Then it is checked if the configuration informations from the service share node is received. If it has not been received yet the the timer is checked again. If the timer then has reached 5 seconds and the

configuration informations has still not been received an error message is shown. If the configurations information has been received the configuration is read and set and a ready message is shown for the user to indicate that the service is ready for use.

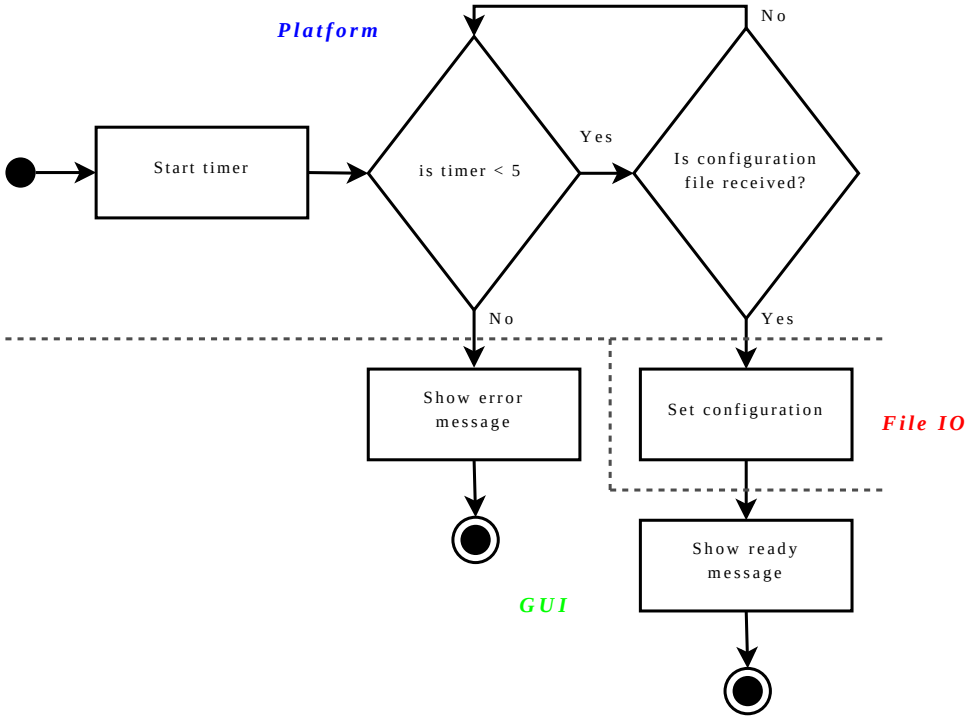

Figure A.3: The receive configuration use case

Figure A.3 contains three different boundaries, File IO, Platform and GUI. Each of the boundaries classifies which activities belongs to which boundaries.

File IO - File input output is for reading the file containing the configuration.

Platform - The Platform starts the timer and checks if the configurations file is received and if the timer is larger than the 5 seconds.

GUI - The GUI shows the error and ready messages for the user.

The activity diagram has two endpoints. The one of the endpoint is reached if an error has occurred during this activity, the other endpoint is reached if the activity is completed successfully.

A.1.2 The service share

It is only possible to share the W3S service if the mobile telephone as 3G net and WiFi capabilities.

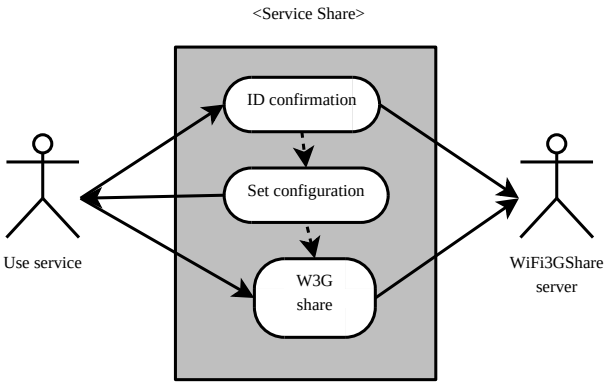

Figure A.4: The service share use case diagram

The Service share is when the user wants to share one or more services. Figure A.4 shows the use cases which are associated with the Service share node from the deployment diagram on figure 4.4. The use case diagram has two actors, the Use service and the WiFi3GShare server. The Use service is the active actor which starts the communication and the WiFi3GShare server is an passive actor which only responds on a requested communication.

ID confirmation

The *ID confirmation* activity is waiting for a connection from a user which would like to use the W3S service. When a connection is made from a use service node the ID the ID conformation activity waits to receive the ID and password from the use service node. When the ID and password is received a connection is made to the WiFi3GShare server. If the connection connection to the WiFi3GShare server is established the ID and password is send to the WiFi3GShare server for confirmation that the ID and password matches a paying subscription, after the reply is received the connection is closed. If the ID confirmation activity is not able to connect to the WiFi3GShare server an error message is send back to the use service node.

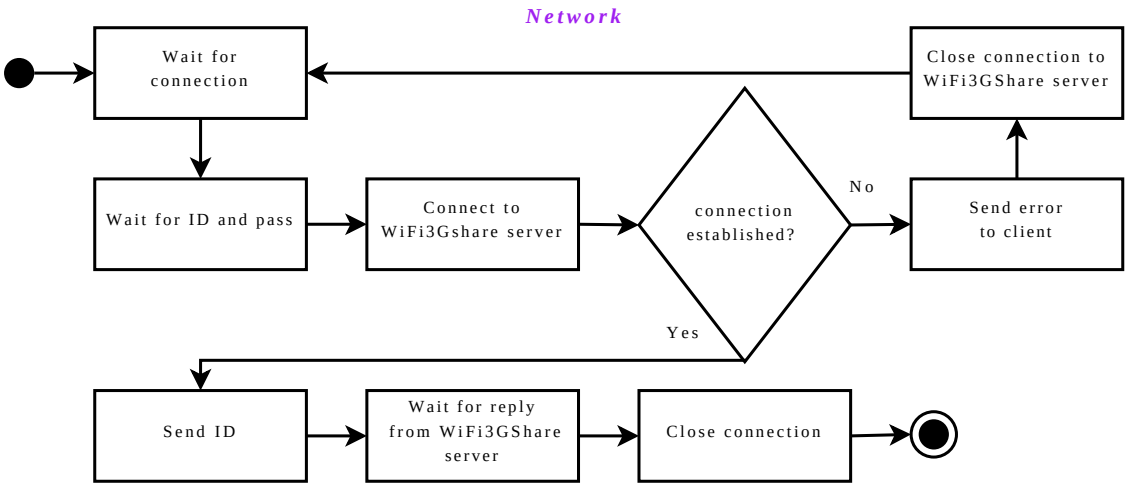

Figure A.5: The ID confirmation use case

Figure A.5 contains only one boundary the Network boundary.

Network - The Network handels all communication over WiFi and 3G net to the WiFi3GShare server.

The activity diagram has one endpoint. The endpoint is reached if the activity is completed successfully.

Set confirmation

The *set confirmation* activity starts when the reply from the WiFi3GShare server is received. If the reply was that the ID and password matched a paying subscription then the amount of points this account has is checked. If the account has an amount of points larger the zero a configuration for 3G is set up. When the configuration is complete a configuration file is made for the use service node, then the file is send to the use service. If the reply from the WiFi3GShare server was that the ID and password was not matched to a paying subscription or the amount of points this account has is zero or less, an error message is send to the use service node and then the connection to the use service node is closed.

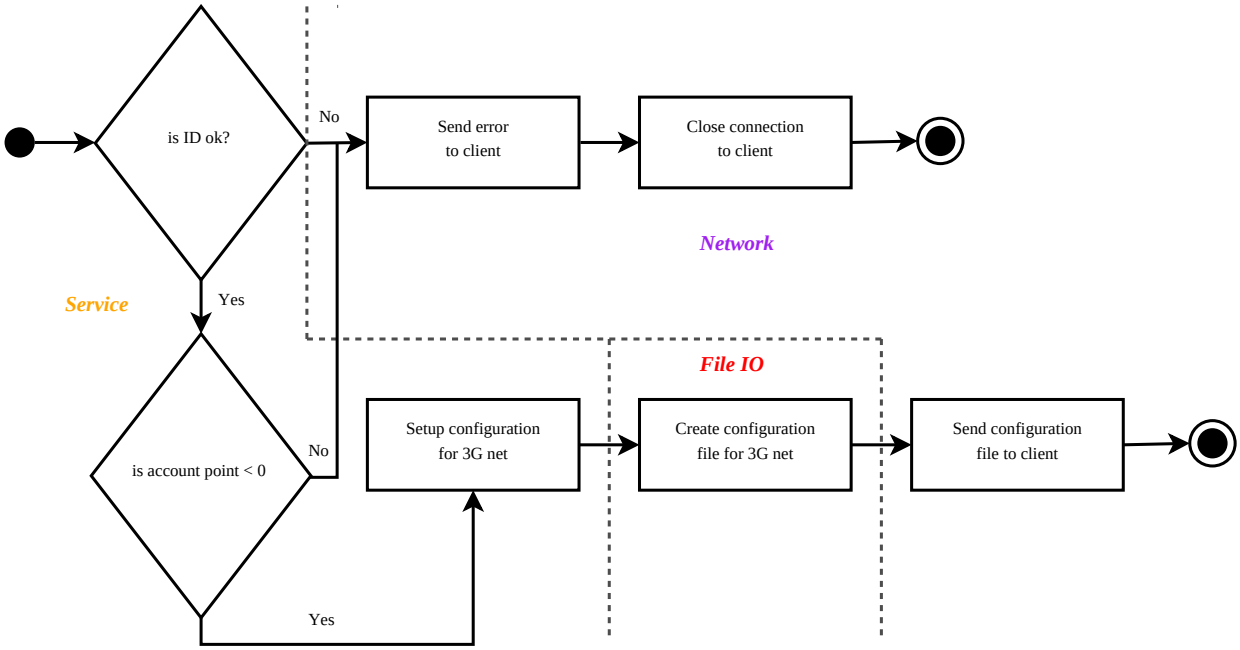

Figure A.6: The set configuration use case

Figure A.6 contains three different boundaries Service, File IO and Network. Each of the boundaries classifies which activities belongs to which boundaries.

Service - The service is the boundary for the W3S which decides based the reply from the WiFi3GShare server if the use service node should have access to the service.

File IO - Creates the configuration file which will be send to the use service node.

Network - Handles the WiFi connection to the use service node.

The activity diagram has two endpoints. The one of the endpoint is reached if an error has occurred during this activity, the other endpoint is reached if the activity is completed successfully.

W3S share

The *W3S share* activity is started when the use service node begins to use the W3S share to connect to the 3G net. When there is connectivity from the use service node the data is monitored. The data monitored is then compared to the points the account has. If the connectivity stops or the monitored data exceeds the amount of points the connection is to the use service node is closed. Afterwards there is made a connection to the WiFi3GShare server and the monitered data and client ID are send.

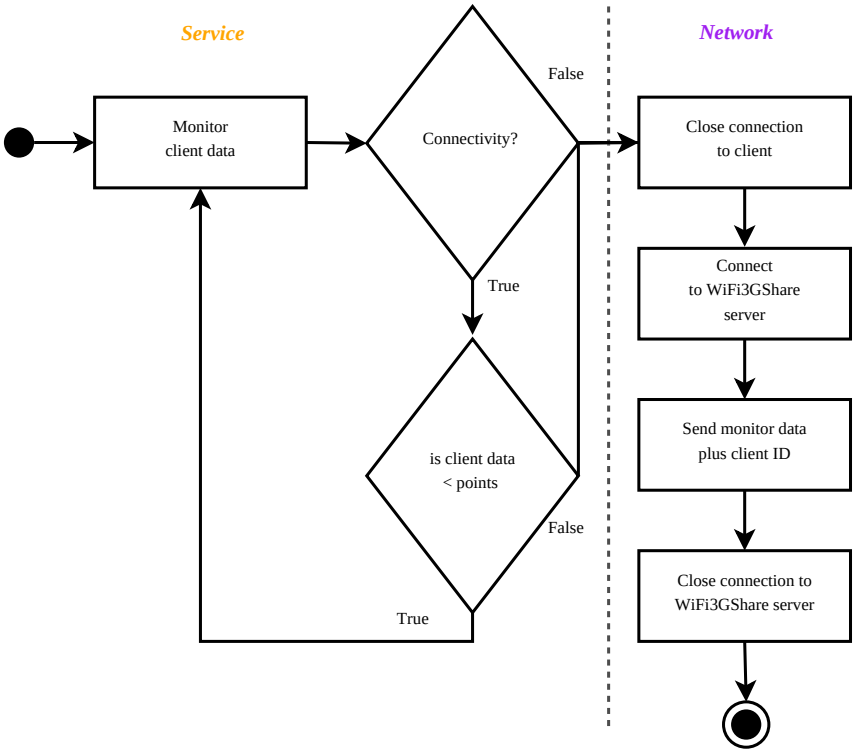

Figure A.7: The W3S share use case

Figure A.7 contains two different boundaries Service and Network. Each of the boundaries classifies which activities belongs to which boundaries.

Service - The Service boundary monitors the use service data traffic.

Network - Handles the 3G connection to the WiFi3GShare server.

The activity diagram has one endpoint. The endpoint is reached if the activity is completed successfully.

A.1.3 WiFi3GShare server

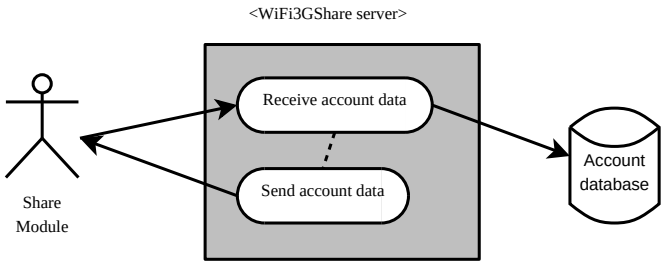

Figure A.8: The WiFi3GShare server use case diagram

The WiFi3GShare server node is where the Share Service node sends an account ID for authentication. Figure A.8 shows the use cases which are associated with the WiFi3GShare server node from the deployment diagram on figure 4.4. The use case diagram have two actors, the Share Module and the account database. The Share Module is the active actor which starts the communication and the account database is an passive actor which only responds on a requested communication.

Receive and send account data

The *receive account data* starts when a Share Service sends account data to the WiFi3GShare server. The data is check if it is a request for authenticating an account or it is monitor data. If it is a request for authentication of an account the account database is read and the ID send from the Share Service is checked if it matches any entries in the account database. When the check is done a reply is send back to the Share Service if the account was authenticated or not. If the account is authentic the amount of points are send as well. If the account data received from the Share Service is monitor data the account database is read and the points used by the Use Service is added to the account which has shared the 3G net.

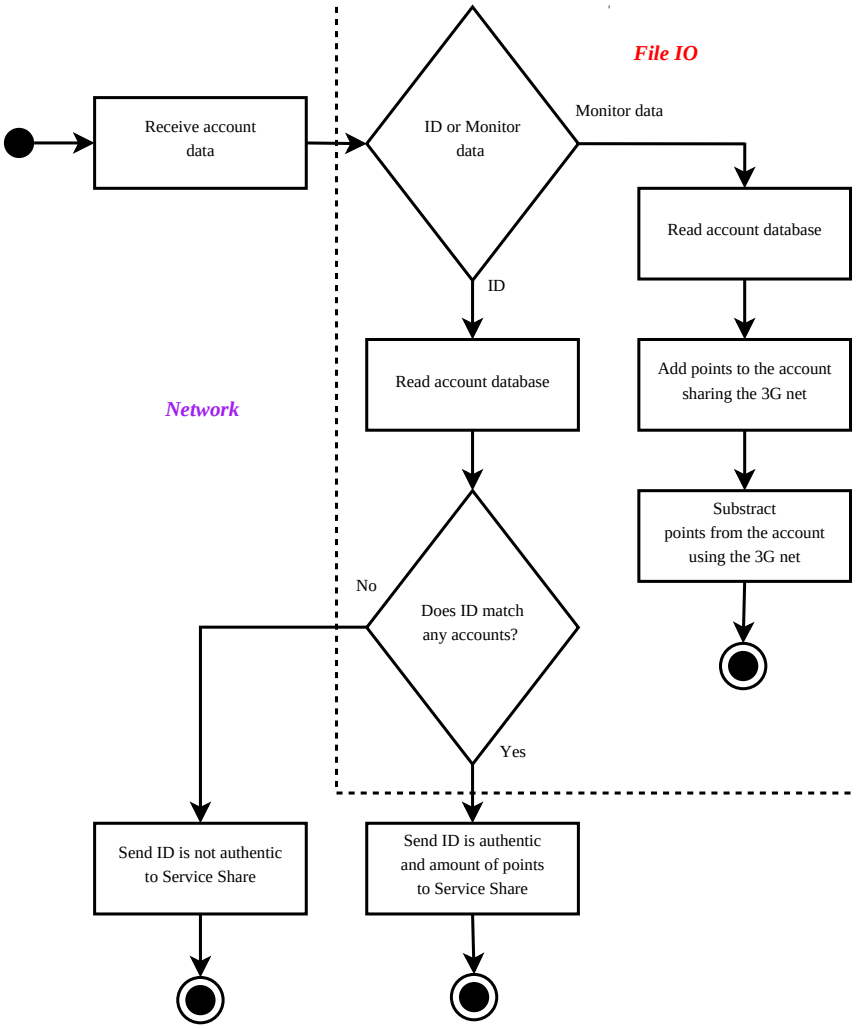

Figure A.9: The receive account data use case

Figure A.9 contains two different boundaries the File IO and Network. Each of the boundaries classifies which activities belongs to which boundaries.

Network - The Network handels all communication over 3G.

File IO - File input output is for reading and writing in the account database.

The activity diagram has three endpoints. the endpoints are reached if the activity is completed successfully.

Literature

[1] T. M. Corporation, “Mobile ad hoc networking,” 2009. http://www.mitre.org/work/tech_transfer/mobilemesh/index.html.

[2] D. Statistik, “Familiernes besiddelse af varige forbrugsgoder fordelt efter forbrugsgoder,” 2008. http://www.statistikbanken.dk/statbank5a/SelectVarVal/Define.asp?MainTable=VARFORBR&PLanguage=0&PXSId=0.

[3] D. Statistik, “Mobil med adgang til internet,” 2008. http://www.statistikbanken.dk/statbank5a/SelectVarVal/Define.asp?Maintable=BEBRIT03&PLanguage=0.

[4] D. Statistik, “Avanceret brug af mobiltelefonen efter type og anvendelsesformï£¡l - pct. af dem der benytter mobiltelefoni,” 2008. http://www.statistikbanken.dk/statbank5a/SelectVarVal/Define.asp?MainTable=BEBRIT06&PLanguage=0&PXSId=0.

[5] D. Radio, “Snak om app stores,” 2009. http://www.dr.dk/P1/harddisken/Udsendelser/2009/07/09140611.htm.

[6] Apple, “Apple app store,” 2009. http://www.apple.com/iphone/apps-for-iphone/.

[7] Nokia, “Nokia ovi store,” 2009. https://store.ovi.com/.

[8] Ingenøren, “Grønne krav til elektronik vinder frem,” 2008. http://ing.dk/artikel/91130-groenne-krav-til-elektronik-vinder-frem?highlight=Gr%F8nne+krav+til+elektronik.

[9] www.sonyericsson.com, “Greenheart mobile from sony ericsson,” 2009. http://www.sonyericsson.com/cws/products/mobilephones/overview/c901greenheart?lc=da&cc=dk.

[10] Greenpeace, “Poisoning the poor electronic waste in ghana,” 2008. http://www.greenpeace.org/denmark/press/rapporter-og-dokumenter/poisoning-the-poor-electroni.

[11] D. Radio, “Genbrugs gadgets,” 2009. http://www.dr.dk/P1/harddisken/Udsendelser/2009/09/09140521.htm.

[12] Telia, “Flexrate/flatrate,” 2010. http://telia.dk/mobil/mobilemuligheder/data/flexrateflatrate/.

[13] UPnP.org, “Upnp device architecture,” 2009. http://www.upnp.org/download/UPnP_Device_Architecture_Generic_20000710.ppt.

[14] S. Cheshire and D. H. Steinberg, *Zero Configuration Networking*. O’REILLY, 2006. ISBN: 0-596-10100-7.

[15] C. P. Subir Kumar Sarkar, T.G. Basavaraju, *Ad hoc mobile wireless networks : principles, protocols, and applications*. Auerbach, 2008. ISBN: 978-1-4200-6221-2.

[16] F. Nokia, “Energy profiler,” 2009. http://www.forum.nokia.com/Technology_Topics/Application_Quality/Power_Management/Nokia_Energy_Profiler_Quick_Start.xhtml.