Where R U?

Project Report

Location Aware Systems

Christer Edvartsen

Institute of Informatics

Østfold University College

Halden, Norway

June, 2006

Østfold University College Mobile Applications Group

Abstract

This report presents a proof of concept which features some basic functionality, and suggestions to future extensions to the project.

Where R U? is a location awareness project can be used to send an automatic response in the form of a map if someone wants to know where you are. It uses the Python for Series 60 platform combined with a bluetooth GPS receiver, and uses an external XML-RPC server to gain access to WMS servers that generate map images. It can answer in the form of an MMS message that is sent from the mobile phone, or as a mail that is sent from the XML-RPC server.

The project comprises a mobile client application written in Python, an XML-RPC server written in PHP and a web based XML-RPC client also written in PHP. The communication between the clients and the server uses the HTTP protocol and the mobile client receives an SMS with some special instructions and automatically fetches the position of the device using a bluetooth GPS receiver. It then proceeds to post the position information along with other instructions from the SMS message to the XML-RPC server. The server will then fetch a map image from a WMS server and send the result to the user who wanted the information in the first place via mail, or back to the mobile phone so the mobile client can send the result as an MMS message.

Keywords: Location Aware Systems, Mobile Applications, Bluetooth, GPS positioning, Python for Series 60, XML-RPC, PHP, WMS

Acknowledgements

I would like to extend appreciation to some people that helped me in some way during this project:

- Gunnar Misund for tutoring me throughout the course of this project
- Glenn Ole Hellekjær for doing his best to teach me how to write reports
- Morgan Jakobsen for providing me with an algorithm for calculating distances between geographical positions

Prerequisites

The implementation part of this report gets quite technical, so knowledge of the PHP and Python programming languages is recommended. The topics are not too complicated, but some code examples might not be straight forward. Knowledge of programming basics such as loops and some basic datatypes are also recommended.

Table of Contents

Chapter 1

Introduction

Mobile phones and devices have become an important part of our everyday lives. The ability to stay in touch with other people at any given time is something that most of us take for granted. In later years the typical mobile phone has become more advanced, enabling the users to do more then just call and send SMS messages. As the devices get more advanced, so do the services and applications that run on these devices. Examples of this might be surfing the Internet, or playing games on the phone. Some phones also make use the the Bluetooth technology that enables the phone to communicate with other devices supporting bluetooth, such as most new laptop computers and GPS receivers. Developers can take advantage of this to make advanced and useful applications for these phones and devices.

This report will describe one oh these, a project called Where R U?[1] that enables users with location aware devices to share information about their whereabouts with each other. This can be done using a mobile phone with bluetooth support running on the Nokia S60 platform. The phone will connect to a portable GPS receiver using the bluetooth specification to gather information of its whereabouts. The next paragraph shows an example on the use of the Where R U? project.

User A is interested in the whereabouts of his/her friend, User B and sends an SMS to User B containing some special instructions. The phone that User B has will then receive the SMS, and when it sees that the message contains these instructions, it will act upon them. The phone will connect to a portable GPS, fetch the position of the device, and send this information to a server along with the original instructions from User A. The server that receives the information from User B will then use a Web Map Service (WMS) to generate a map pinpointing the location of User B. This map will then be sent in a format that is explained in the original message that User A sent.

[1]The name of the project is written using the textual language that is often used when writing SMS messages.

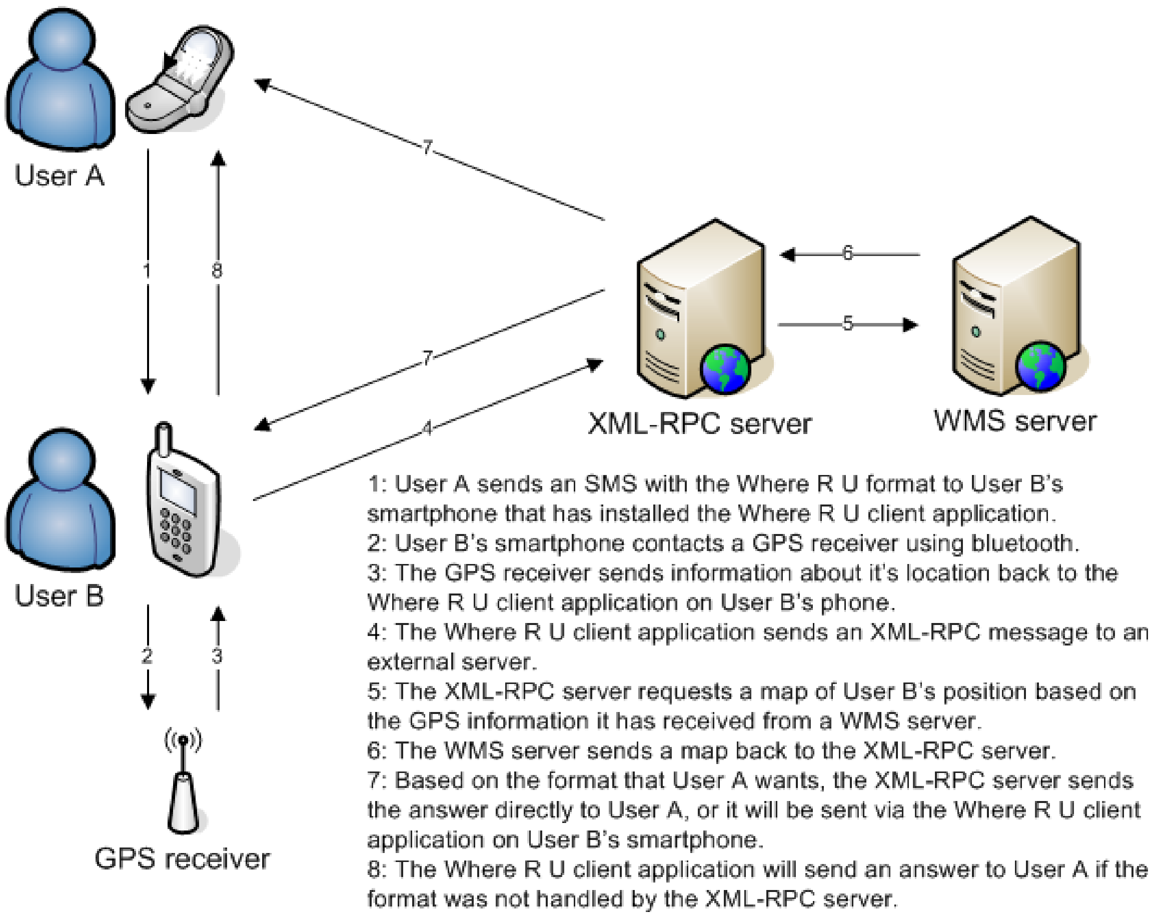

User A
User B
GPS receiver
XML-RPC server
WMS server
1
2
3
4
5
6
7
7
8
1: User A sends an SMS with the Where R U format to User B's smartphone that has installed the Where R U client application.
2: User B's smartphone contacts a GPS receiver using bluetooth.
3: The GPS receiver sends information about it's location back to the Where R U client application on User B's phone.
4: The Where R U client application sends an XML-RPC message to an external server.
5: The XML-RPC server requests a map of User B's position based on the GPS information it has received from a WMS server.
6: The WMS server sends a map back to the XML-RPC server.
7: Based on the format that User A wants, the XML-RPC server sends the answer directly to User A, or it will be sent via the Where R U client application on User B's smartphone.
8: The Where R U client application will send an answer to User A if the format was not handled by the XML-RPC server.

Figure 1.1: Information flow in the Where R U? project

This might be an MMS message, mail or some other format. Figure 1.1 illustrates this information flow.

Above I have explained the main idea behind the Where R U? project. The rest of this report will include more detailed information on how the project works, and on which devices it will work.

This project will demonstrate how easy the Python for Series 60 platform can be used to make useful and interesting software. It will also show how easy it is to have different applications talk to each other regardless of the language the each application is written in.

Chapter 2 provides the readers the necessary background information on the different technologies and methods used in this project. It also contains information on the documentation used when developing the project. Chapter 3 explains the design of the project in further detail, and explains some of the design problems and technology shortcomings that I encountered. Chapter 4 goes into detail on how the different parts of the project was implemented. Chapter 5 will include the results of the project followed by chapter 6 that deals with the testing of the software produced in this project. The last chapter goes into further discussion about what might be done differently, some ideas on how to expand the project and conclusions.

Chapter 2

Background

This chapter will provide background information on the different technologies and methods used in the Where R U? project. First we start with the technology part, and then we will give some information regarding the methods used. Last we will look at some of the documentation used in the project.

2.1 Technologies

2.1.1 Location Aware Systems

Computer systems that "behave" in different ways depending on their location/context are called Location Aware Systems. An example of a Location Aware System is a Global Positioning System (GPS). A GPS receiver fetches information from satellites that contains the position of the GPS receiver. This project will then use a GPS connected to a mobile phone using the bluetooth specification to get the position of the mobile phone. After the Where R U? application that runs on the phone receives the position data, it can act on these, making it a location aware system.

2.1.2 Smartphones

A smartphone is an electronic handheld device that integrates both the functionality of a mobile phone, a personal digital assistant (PDA) or some other information appliance. Some of the most popular PDA functions include address books, calendars and task lists.

One of the most important features of a smartphone is the possibility to install additional software. This enables developers to write their own software for the smartphones. The most common operating systems that run on these devices are Symbian, Palm OS, Windows Mobile, BREW and Linux [1]. Since smartphones include more advanced features they tend to be more powerful than regular mobile phones. Because of this, developers can come up with more advanced features in their software. Handling graphics (i.e. Digital Maps) and audio/video is no longer that hard to do because of the increased performance of the devices.

2.1.3 Web Map Service

A Web Map Service (WMS) is a service that generates map images. These images can be in a format such as JPEG, PNG, GIF or Scalable Vector Graphics (SVG). A WMS only returns the map as an image, as opposed to a Web Feature Service (WFS) that delivers the actual data that can be used to generate an image.

Retrieving a map from a WMS can be done by simply entering an URL in an internet browser. The content of the URL will inform the WMS what the map it is supposed to generate will contain. One can decide on which format to use, the width and height of the image, and different layers. The layers can for instance be water, roads, buildings, and road names. Figure 2.1 shows an URL that retrieves an image from Statens Kartverk [2].

Figure 2.1: Retrieving an image from a WMS

```
http://basiswms.statkart.no/servlet/com.esri.wms.Esrimap?WMTVER=1.0&REQUEST=
map&SRS=EPSG:4326&FORMAT=PNG&BGCOLOR=0x23f3f5&TRANSPARENT=TRUE&STYLES=default
,,,,,,,,,,,,,,,&LAYERS=Landtone,Vann,Markslag,Vannkontur,Elv,Jernbane,
Kommunegrenser,Hoydekurver,Bygninger,Bygninger_grunnriss,N50Bebyggelse,
Bebyggelse,Adresser,Gatenavn,Stedsnavn,Fylkesnavn,Veg&ServiceName=
Topografisk_Norgeskart_wms&BBOX
=11.3643740046,59.1144180961,11.4139785439,59.1398719039&WIDTH=800&HEIGHT=600
```

When the WMS receives a request such as the one in figure 2.1, it will generate a map image containing the layers specified in the request, and the area specified by the bounding box in the BBOX argument in the URL. The first two coordinates in the bounding box are the longitude and latitude positions of the lower left corner. The last two coordinates are then the upper right corner of the bounding box. The WMS will return a PNG image with a width of 800 pixels and a height of 600 pixels. Most WMS servers have implemented a method of gaining information on what the WMS supports called GetCapabilities. Statens Kartverks WMS will give this information if the URL in figure 2.2 is used. According to the WMS specification [3] by the Open Geospacial Consortium (OGC), all WMS servers must implement GetCapabilities. As stated in the specification, GetCapabilities should generate a machinereadable (and human-readable) description of the servers information content and acceptable request parameter values.

Figure 2.2: Retrieving the capabilities of a WMS

```
http://basiswms.statkart.no/servlet/com.esri.wms.Esrimap?WMTVER=1.0&REQUEST=
GetCapabilities&ServiceName=Topografisk_Norgeskart_wms
```

2.1.4 Nokia S60 platform

Nokia S60 (formerly Series 60) is a software platform for smartphones with advanced data capabilities and is optimized for the Symbian OS. It is developed primarily by Nokia. A set of standard applications and libraries comprises the S60 platform. Some standard applications are for instance telephony and PIM tools. According to [4], Nokia S60 is amongst the leading smartphone platforms in the world.

Nokia S60 has been implemented on many mobile phones, and some of them are: Panasonic X700, Nokia 6620, Nokia 6680 and Nokia N70. This is a small sample of the phones that support this platform.

Developers can work in C++ (using Symbian OS' native APIs), the Java language, or Python. The latter is the language of choice for this project and more information about Python can be found in section 2.1.7 in this chapter.

More than 25 million S60 platform devices had been shipped by May 2005 [5].

2.1.5 Global Positioning System

The Global Positioning System (GPS) is a satellite navigation system. More than two dozen GPS satellites [6] broadcasts timing signals by radio to GPS receivers, allowing them to accurately determine their location (longitude, latitude, and altitude). This project uses a GPS receiver to pinpoint the location of the mobile phone. The longitude and latitude values fetched from the GPS receiver will eventually be embedded in the URL that is sent to the WMS server. The GPS that the Where R U? project uses is a Holux GPS receiver as pictured in figure 2.3. The mobile phone and the GPS receiver will use the Bluetooth technology for communication (more about Bluetooth in the next section).

Figure 2.3: Holux GPSlim 236 GPS receiver

2.1.6 Bluetooth

Bluetooth is a specification for wireless personal area networks, also known as IEEE 802.15.1. Using bluetooth, devices can connect and exchange information between each other. Both the Nokia 6680, and the Holux GPS receiver has bluetooth support.

2.1.7 Python

As stated on the Python homepage [7]:

> Python is a dynamic object-oriented programming language that can be used for many kinds of software development. It offers strong support for integration with other languages and tools, comes with extensive standard libraries, and can be learned in a few days. Many Python programmers report substantial productivity gains and feel the language encourages the development of higher quality, more maintainable code.

One of the main reasons for using Python in the Where R U? project is because I have had more experience with that as opposed to C++ and Java. If one of the other programming languages were to be used, a lot of time would go by to just learn the languages better, and the project would probably not have come very far.

The mobile client is the part of the project where Python is used. The XML-RPC server could be written in Python as well, but that part ended up using PHP instead. More about that in section 2.1.8.

Python for Series 60

Python for Series 60 is an implementation of the 2.2.2 version of the Python interpreter for the Nokia S60 platform. The implementation includes a selection of the standard Python modules, a shell for the launching of Python scripts, a variety of native extensions, and a Python Console for interactive development.

Figure 2.4: Nokia 6680 mobile phone

At this moment no official MMS module exist in the Python for Series 60 project. The Where R U? project was first supposed to send MMS messages from the mobile phone. This could not be done since no MMS module was available. The MMS handling was then moved to the XML-RPC server. Late in the project, an unofficial MMS module [8] was released that could be used to send MMS messages from the mobile phone instead, and the MMS handling was again moved to the mobile client.

The main reason for an official MMS module not being implemented yet is that the developers of Python for Series 60 have given other modules higher priority [9]. Since Python for Series 60 is open source, developers can easily write their own modules for Python for Series 60 using the native Symbian OS C++ APIs. One option in the early stages of the project was to try and implement an MMS module, but that would probably have taken too much time.

The Nokia 6680 phone, as shown in figure 2.4, will be used for developing and testing in the Where R U? project. The phone uses Nokia S60 2nd Edition, Feature Pack 2 (Version 2.6) and uses the 8.0a version of the Symbian OS.

2.1.8 PHP

The XML-RPC server and the web-based test client was written using the PHP programming language. As mentioned above this is because PHP is the language I have had the most experience with, so choosing PHP saved me the time of learning how to do these parts of the project in Python. The following is a quote from the official PHP webpage [10]:

> PHP is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML.

As we can see, it states that PHP is especially suited for Web development, for which it will be used in this project.

2.1.9 XML-RPC

XML-RPC is a remote procedure call protocol. It uses XML to encode its calls and HTTP as a transport mechanism. With XML-RPC, a client can call methods with parameters on a remote server, and get the results in a structured form. The protocol is fairly simple, defining only a few datatypes and commands.

The Where R U? XML-RPC server uses an open source library called phpxmlrpc [11]. This server will receive requests from the mobile application written in Python, and the web-based test client written in PHP. The test client uses the same XML-RPC library to encode the request that it sends to the server. Once the server receives a request, it will do what it is told, and send a result back to the client.

The Where R U? mobile client uses a Python library called xmlrpclib [12] that do the same for Python as the client part of the phpxmlrpc library do for PHP.

2.2 Methods

Some of the technologies used in the Where R U? project were new to me when I started the project in January 2006. To be able to finish such a project in a relative short timespan, I used the Extreme Programming methodology.

Extreme Programming (Sometimes referred to as XP) is a software engineering methodology. It is especially useful in a project such as Where R U? since it prescribes a set of day-to-day practices. Extreme programming was developed by Kent Beck, who published the first book about the methodology in 2000. The method features twelve practices for the development of a project. Some, but not all, of these have been used in the Where R U? project. The twelve practices of Extreme Programming can be found in appendix B and as we can see from them, they fit better for a larger group of people, but can also be used when there is only one person in the group, as is the case with this project.

2.3 Developing and testing

The development of the XML-RPC server and the web-based test client was done on the Linux platform. The reason for this is that both these parts of the project will be run on a Linux server. They could have been developed on the Windows platform as well, but it is easier to replicate the production environment using the same platform when developing.

The Where R U? mobile client, however, has been developed on the Windows platform using a regular text editor to write code in, and the Symbian OS emulator to test the applications. It can save developers a lot of time if they use an emulator for testing instead of transferring their applications to a mobile phone to test it there. However, there are some parts of testing in the emulator that might be a bit tricky, for example getting the bluetooth GPS receiver to work together with the emulator.

2.4 Available Documentation

2.4.1 Online documentation

Both PHP and Python have good online documentation which can be found on [10] and [7] respectively.

2.4.2 Books about programming in Python

There are many books about the Python programming language on the market, but two where used in this project, mainly because I already had these books when starting the project. Since the Python for Series 60 interpreter uses regular Python, one does not need to learn a “new” Python language.

The two books used are Learning Python [13], and Programming Python [14]. Experienced Python programmers would not need to use books on Python, but rather concentrate on the modules that exist in Python for Series 60. When you download the Python for Series 60 package you will also get the Python for Series 60 API documentation [15] which will describe all the different modules available.

2.4.3 Online discussion boards

Nokia provides developers with online discussion boards that enables them to exchange information and knowledge. This can be very helpful when dealing with new technology. Users can search the forums for information they want, and if it can not be found, one can simply post a question there and wait for other users to answer.

The developers of Python for Series 60 at Nokia also use the discussion boards. They have made some posts asking other developers which features they would like to be included in the next version or Python for Series 60. Some of the wishes that other developers have mentioned has become available in newer versions of Python for Series 60. The inbox module that will be discussed in chapter 4 is an example of this. This is a great opportunity for the developers at Nokia to see what other developers want, so they can make Python for Series 60 a better platform.

2.5 Summary

Numerous technologies have been used in this project, and having read this chapter you should posses a good overview of how the different parts of the project have been developed. A technology such as XML-RPC helps the developer combine different parts of a project into one “application”. It does not matter what programming language have been used on the different parts since they speak to each other using the XML-RPC standard.

This chapter has also provided information on the documentation used in this project. Good documentation is very important when developing a project that not to many others have done. One of the most important sources used for the Where R U? project is the online discussion boards provided by Nokia where professionals from Nokia along with regular users help each other.

The next chapter will go into detail on how the different parts of this project have been designed.

Chapter 3

Design

The Where R U? project consist of three parts; the XML-RPC server, a web based test client and the mobile client application. This chapter deals with the design of these parts, and how they have evolved from the start of the project. First we will look at an example scenario where the project could be used, and then the design of the different parts of the project will be presented.

This chapter only covers the design of the parts of this project. How they have been implemented is presented in chapter 4.

3.1 Example scenario

A project like Where R U? can be used in a number of scenarios. One scenario is explained in the next section.

This illustrates why the mobile client application has been designed so that very little user interaction is needed for it to work. This could prove very useful when someone is lost and may not be able to answer the phone.

3.1.1 Search and rescue

Richard is on a mountain climbing trip and sets out alone to gather some information about a couple of new climbing routes. Before he sets out he starts the Where R U? application and brings along his bluetooth GPS receiver. He informs his friends that he will be back in a couple of hours. As the day goes by Richard does not return and his friends are getting anxious. They try to call Richard but he does not answer. The next thing they do is send a Where R U? SMS to Richard, and receive an answer via MMS shortly after containing a map that informs them where Richards mobile phone is. They set out looking for him, and quickly find him. It turns out that Richard had slipped on the rock and hit his head. He was unconscious and could not operate his mobile phone.

3.2 XML-RPC server

The XML-RPC server used in this project is small and easy to expand. It uses the phpxmlrpc library that is designed for ease of use, flexibility and completeness, which fits the Where R U? project perfectly. The library is made in such a way that it is very easy to implement more methods later on. There are only two methods implemented at the moment, and adding more later should not be a problem.

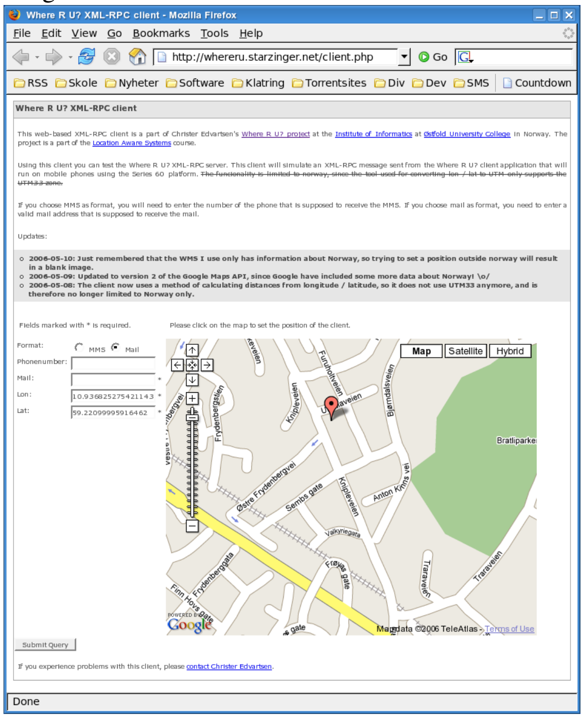
When a user requests something from the server, a pair of coordinates is sent that will be the center point of the map. The XML-RPC server will then use a WMS to generate a map image. WMS servers need a bounding box to be able to generate images, and since we only have the center point of the box, we need to calculate two of the corners of the box. At first, the center point was converted to UTM using a service available at the OneMap site [16] because we wanted to specify a radius in meters. UTM is a grid-based method of specifying locations on the surface of the Earth and is easier to work with than longitude and latitude when using distances specified in meters. The UTM grid consist of several "zones" and each zone uses different conversion methods to convert from longitude and latitude. [16] only supported the zone used in norway and therefore limiting the XML-RPC server to only generate correct values when the centerpoint was in Norway. The UTM conversion was later discarded, and the server ended up calculating with longitude and latitude values instead.

3.3 XML-RPC test client

The sole purpose of the test client was simply to test the XML-RPC server. A screenshot of the client can be seen in figure 3.1.

As we can see from the screenshot, the user have the possibility to choose MMS or mail as a medium. In the early stages of the project, the XML-RPC server was supposed to be able to generate MMS messages. This was later changed since an unofficial MMS module was release for Python for Series 60. Instead of removing the MMS option from the test client, the XML-RPC server returns an error informing the client that the MMS medium is not implemented. One reason for this was so the MMS module could more easily be implemented later, since no changes would need to be made

Figure 3.1: Screenshot of the XML-RPC test client


Where R U? XML-RPC client - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://whereru.starzinger.net/client.php
Go
RSS Skole Nyheter Software Klatring Torrentsites Div Dev SMS Countdown
Map Satellite Hybrid
Bratliparken
Done

Where R U? XML-RPC client

This web-based XML-RPC client is a part of Christer Edvartsen's Where R U? project at the Institute of Informatics at Østfold University College in Norway. The project is a part of the Location Aware Systems course.

Using this client you can test the Where R U? XML-RPC server. This client will simulate an XML-RPC message sent from the Where R U? client application that will run on mobile phones using the Series 60 platform. ~~The funcionality is limited to norway, since the tool used for converting lon / lat to UTM only supports the UTM33 zone.~~

If you choose MMS as format, you will need to enter the number of the phone that is supposed to receive the MMS. If you choose mail as format, you need to enter a valid mail address that is supposed to receive the mail.

Updates:

- **2006-05-10: Just remembered that the WMS I use only has information about Norway, so trying to set a position outside norway will result in a blank image.**
- **2006-05-09: Updated to version 2 of the Google Maps API, since Google have included some more data about Norway! \o/**
- **2006-05-08: The client now uses a method of calculating distances from longitude / latitude, so it does not use UTM33 anymore, and is therefore no longer limited to Norway only.**

Fields marked with * is required. Please click on the map to set the position of the client.

Format:	MMS [✓ Mail]
Phonenumber:	
Mail:	*
Lon:	10.936825275421143 *
Lat:	59.22099995916462 *

Submit Query

If you experience problems with this client, please contact Christer Edvartsen.

in the test client which is ready to request MMS messages.

3.4 Where R U? SMS specification

The SMS specification was designed to be very easy to use. The message must start with “whereru” and then followed by a medium, and the recipient of the request. More options can later be added in the SMS making it more advanced, giving the users more options.

3.5 Mobile client application

The client application was meant to be used without any interaction from the user. This meaning that the user could start the application, and just let it run in the background so that people could send requests to the phone. There are some problems related to this. On the Nokia S60 platform, the user can set some application preferences that can let the application connect to the internet whenever it wants without any interaction from the user. This can not be done for Python for Series

60 applications. Whenever the application is supposed to send an XML-RPC request or send an MMS the user will have to confirm the connection. This will only happen once for each time the application is started, so it can easily be circumvented.

3.6 Summary

When using many new technologies that are under constant development it is not easy to set definite design goals early in the project. New feature become available during the development of the project that might help you in some way, or enable you to do something that was previously not possible. The MMS module used in this project is a good example of that, and this is something one can expect when working with new technologies.

As mentioned earlier, the next chapter will go into detail on the implementation of the different parts of the project.

Chapter 4

Implementation

The following sections will describe how the different parts of the Where R U? project has been implemented. First we will look at the XML-RPC server that handles requests from mobile clients. Then we will look at a web-based test client that was made prior to the mobile client to see if the server worked as intended. Then we move on to the SMS specification used in the mobile client, and last we will look that the mobile client itself.

4.1 XML-RPC server

The XML-RPC server is the part of the project that does most of the work. It receives requests from Where R U? clients, parses this and fetches a map from a WMS, and sends this to the person who wanted it in the first place. The server is written in PHP using the phpxmlrpc library. phpxmlrpc is designed for ease of use, flexibility and completeness. The library is not the fastest, but that is not considered an important factor of this project.

Figure 4.1 shows the few lines that is necessary to set up the server using the phpxmlrpc library.

The constructor of the xmlrpc_server class takes an array as argument. The array will define the methods available from the server. This server has two methods available: "find_me" and "fetch_map". Each method is defined using another array. That array consist of three elements: "function", "signature" and "docstring". The two methods available will both be discussed in detail in section 4.1.1.

"function" is the name of the PHP function that will be executed when an XML-RPC client requests one of the methods available. In this case a function called "xmlrpc_find_me" will be executed if a client requests the find_me method.

Figure 4.1: Setting up the XML-RPC server

```
$server = new xmlrpc_server(array(
  'find_me' => array(
    'function' => 'xmlrpc_find_me',
    'signature' => array(array($xmlrpcBoolean, $xmlrpcArray)),
    'docstring' => '',
  ),
  'fetch_map' => array(
    'function' => 'xmlrpc_fetch_map',
    'signature' => array(array($xmlrpcString, $xmlrpcArray)),
    'docstring' => '',
  ),
));
```

"signature" contains information on the input and output of the methods. This is defined using an array where the first element is the output, and the remaining elements is the arguments of the method. These must again be put in an array, since a method can have several signatures. As we can see in figure 4.1, both methods have the same input, but different output. find_me has an $xmlrpcBoolean as output, and the fetch_map method have an $xmlrpcString as output. The client that will be discussed in section 4.4 is written in Python, which uses lists instead of arrays. The phpxmlrpc library will take care of the input from the client for us, so we do not need to convert the lists to arrays ourselves.

The last element to include when setting up the methods on the server is the "docstring" element. This element is just a regular string that can contain information on how the method works.

4.1.1 XML-RPC server methods

This section will explain how the two methods implemented on the server work. First we start with the find_me method, and then we move on to the fetch_map method.

find_me

When using this method, everything will be done on the server, and a boolean will be sent back to the client to inform it on how the operation went. This method is currently only used when sending the answer to a mail address. Since we can not generate an MMS message and send it through the server, this method can not be used for MMS. The fetch_map method explained in the next section will be used for MMS.

As we have seen this method executes the xmlrpc_find_me PHP function that takes an array as an argument. The first thing that xmlrpc_find_me does is to fetch the arguments inside the array. As we will see in the section about the web-based client, the arguments we receive, contains two doubles and two strings. The arguments are:

longitude (double)
The longitude part of the position of the client.

latitude (double)
The latitude part of the position of the client.

format (string)
The format to use. This might be "mail" or "MMS".

recipient (string)
The recipient. If format is "mail" this is a mail address. If format is "MMS" this is a phone number.

img_type (string) (optional)
Type of the image. Can be "png" or "jpg". Defaults to "png".

width (integer) (optional)
Width of the image to be generated in pixels. Defaults to 800.

height (integer) (optional)
Height of the image to be generated in pixels. Defaults to 600.

Once it has fetched the arguments, it checks if the longitude and latitude values are valid. If they are not, the server returns an error. The way this is done in phpxmlrpc can be seen in figure 4.2.

Figure 4.2: Returning an error using phpxmlrpc

```
return new xmlrpcresp(0, 804, WRU_804);
```

When returning an error, a new instance of the xmlrpcresp class is used. When passing 0 as the first argument, the class understands that we have an error. The second argument is the error code. According to the phpxmlrpc documentation [17], user errors start at 800. The last argument is the textual representation of the error. As we can see in 4.2 a constant is used. The different error codes that the Where R U XML-RPC server returns can be found in table 4.1 in section 4.1.2.

After the function has validated the longitude and latitude values, it will see what format the message is to be sent in. Since mail is the only format supported by the find_me method, all other formats will return an error to the client. After it has figured out that the user wants the result sent to a mail address, it will see if the mail address is a valid address. This is done using a simple regular expression. If the mail address is invalid, an error is returned. If not, the server will continue its execution of the function.

Next is to try and fetch a map from a WMS using the same longitude and latitude arguments it received from the client. The WMS used in this project is Statens Kartverks' WMS. It only contain data about Norway, so if the client sends longitude and latitude values that lie outside of Norway, the WMS will only return a blank image. However, since most WMS use the same standards it should not be any problem to change the WMS later. To fetch the map image from the WMS, another PHP function is used: "get_map_image". This function is called with the longitude and latitude as arguments, along with a radius, and a variable that will be set to the URL of the WMS request. Since the longitude and latitude values are the centerpoint of where the client is positioned, the radius can be used to determine how detailed the map will be.

To get a map back from a WMS, we will need to make a "bounding box". The WMS will then fill this box with a map image. Since we only have the center of the box, and the WMS needs the bottom left and top right corner to generate a map, we will need to calculate the longitude and latitude of the two corners. A class called gPoint was made that stores information about one geographical point, and it has a method that can calculate the positions of the two corners we need. The method used can be seen in figure 4.3.

Figure 4.3: Calculating the distance of two geographical positions

```php
public function moveTo($bearing, $range){
  $angle = deg2rad($bearing); // convert degrees to radians
  $dist = $range / (1852 * 60); // 1852 = meters pr. nautical mile
  $res = new gPoint(0, 0); // make new gPoint object
  $res->setLat($this->lat + $dist * cos($angle));
  $midLat = ($this->lat + $res->getLat()) / 2;
  $res->setLon($this->lon + ($dist / cos(deg2rad($midLat))) * sin($angle));

  return $res;
}
```

We will not go into detail here on how this function works. The function will use the position of the object is is executed from, and return a new gPoint object at the given range and bearing from that object. Figure 4.4 show how we get the two corners.

Figure 4.4: Calculating the corners of a bounding box

```
$gp = new gPoint($lon, $lat);

$bl = $gp->moveTo(225, $rad); // bottom left
$tr = $gp->moveTo(45, $rad); // top right
```

First we make a gPoint using the position sent from the client. $bl will contain the position at the bottom left of the bounding box, and $tr will contain the position at the top right. As we can see, bottom left is positioned at an angle of 225 degrees from the center, and top right is 45 degrees from the center. $rad contains the amount of meters we will move, which in this case is 2000. $bl and $tr can then be used in the request to the WMS. The complete request can be seen in figure 4.5.

Figure 4.5: WMS request

```
$source = 'http://basiswms.statkart.no/servlet/com.esri.wms.Esrimap?WMTVER
=1.0&REQUEST=map&SRS=EPSG:4326&FORMAT=PNG&BGCOLOR=0x23f3f5&TRANSPARENT=TRUE&
STYLES=default,,,,,,,,,,,,,,,&LAYERS=Landtone,Vann,Markslag,Vannkontur,Elv,
Jernbane,Kommunegrenser,Hoydekurver,Bygninger,Bygninger_grunnriss,
N50Bebyggelse,Bebyggelse,Adresser,Gatenavn,Stedsnavn,Fylkesnavn,Veg&
ServiceName=Topografisk_Norgeskart_wms&BBOX=' . $bl->getLon() . ',' . $bl->
getLat() . ',' . $tr->getLon() . ',' . $tr->getLat() . '&WIDTH=800&HEIGHT=600
';
```

The URL informs the WMS that we use EPSG:4326 projection which uses longitude and latitude, and it will also tell the WMS which layers it shall use in the image it returns. As we can see from the URL, the image returned will be a PNG image that is 800 x 600 pixels. The last part of the URL contain the bounding box, and here we can see the the $bl and $tr objects in use. After the image is fetched from the WMS, it is saved on the server so it can be sent as an attachment to a mail address later on. Before it is sent, the server will do a simple modification to the image. It will place a marker at the center of the image to pinpoint the position of the client. To do this, a version of the GD [18] library that is bundled with PHP is used. One way to do this can be seen in figure 4.6.

$dest and $marker contain the filenames of the original image and the marker image respectively. First we need to make image resources out of the two images, and since both are PNG images, we will use the imagecreatefrompng function available in PHP. When that is done, we need to know the dimensions of the two files to be able to place the marker in the middle of the map. This is done using the getimagesize function. The function returns an array, and as we can see, the first element in the array is the width of the image, and the second element is the height. The array contains more information as well, but we do not need more here. To merge the two images the

Figure 4.6: Placing a marker on a map image

```
$dest = 'original.png';
$marker = 'marker.png';

$dest_im = imagecreatefrompng($dest);
$marker_im = imagecreatefrompng($marker);

$dest_size = getimagesize($dest);
$dest_width = $dest_size[0];
$dest_height = $dest_size[1];

$marker_size = getimagesize($marker);
$marker_width = $marker_size[0];
$marker_height = $marker_size[1];

imagecopymerge($dest_im, $marker_im, (($dest_width / 2) - ($marker_width / 2)
), (($dest_height / 2) - ($marker_height / 2)), 0, 0, $marker_width,
$marker_height, 100);

imagepng($dest_im, $dest);
```

imagecopymerge function is used.

After the images have been merged, the new image is saved at the location of the old image, replacing the old one. The name of the destination file will then be returned from the get_map_image function. If an error occurs in the get_map_image function it will return false instead of the filename. The xmlrpc_find_me that called up get_map_image will see if the return value is false or not. If false, it will return an error to the client, informing it that an error occurred. If get_map_image returned a filename, xmlrpc_find_me can continue its execution. The only part left is to actually send the mail to the person who wanted the map image in the first place, using a class called phpmailer. How this is done in the XML-RPC server can be seen in figure 4.7.

The phpmailer class is very easy to use, and it enables the user to easily attach files to outgoing mails. First we need a phpmailer object. After the object is created, some attributes are set. We can for instance set the mail address of the sender. In this case it will look like christer.edvartsen@hiof.no sent the mail. The AddAddress method adds a recipient to the mail. $recipient is in this case the mail address sent from the client. The last thing to do before the mail is sent is to add the map image as an attachment. That is done using the AddAttachment method, which can take two arguments. The second argument is optional, and if set it will be the filename of the attachment that will appear in the mail client that receives the mail. In this case the filename will be for instance “whereru_20060509212902.png”. The numbers in the filename refers to the time on

Figure 4.7: Sending a mail using the phpmailer class

```
$mail = new PHPMailer();

$mail->From = 'christer.edvartsen@hiof.no';
$mail->FromName = 'Where R U?';
$mail->AddAddress($recipient);
$mail->Subject = 'Where R U?';
$mail->Body = "URL used to generate map:\r\n" . $source . "\r\n\r\n";
$mail->AddAttachment($filename, 'whereru_' . date('YmdHis') . get_extension(
$filename));

if($mail->Send()){
  return new xmlrpcresp(new xmlrpcval(true, 'boolean'));
} else{
  return new xmlrpcresp(0, 802, WRU_802);
}
```

the server when the mail is sent. The format of the time is: year, month, day, hour, minute, second.

To send the mail, the Send method is used. If the mail is sent from the server, the Send method returns true, and the function will return true to the client. If the mail is not sent, an error will be returned.

fetch_map

This method fetches a map to the server, and sends the URL of the fetched map back to the client. The reason that we do not do everything on the client is that we might want to do some changes to the original map before using it in the client. This is better to do on the server side, since image manipulation might take a while on smaller devices.

This method does almost the same as the find_me method, but instead of checking for which medium to send, it will send an URL back to the client containing the address to the map image. This method uses the same PHP functions as find_me to validate the longitude and latitude values and fetch the map image from the WMS.

4.1.2 XML-RPC server error codes

Table 4.1 contain the different error codes that the Where R U? XML-RPC server can return, along with their textual representation.

The reason that the error codes start at 800 is that the phpxmlrpc library has reserved all codes below 800 [17].

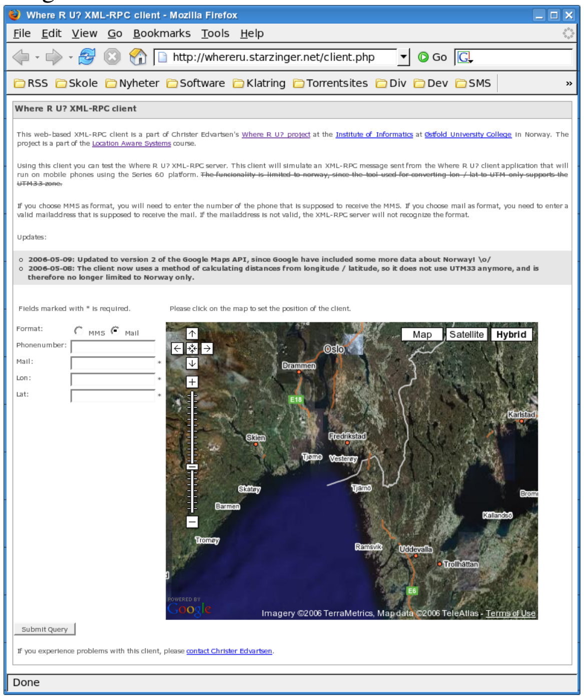
Table 4.1: Where R U? XML-RPC server error codes

Error code	**Error message**
800	Format not implemented.
801	Could not fetch map image.
802	Could not send mail.
803	Unknown format.
804	Illegal longitude / latitude values.
805	Invalid mail address.

4.2 XML-RPC test client

When the server was partially finished, a client had to be made to test the server. Instead of starting directly on the mobile client, a simple web-based client was made. It was written in PHP using the same phpxmlrpc library that was used for the server. Figure 4.8 shows a screenshot of the client as seen in the Mozilla Firefox web-browser.

Figure 4.8: Screenshot of the XML-RPC test client


Where R U? XML-RPC client - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://whereru.starzinger.net/client.php
Go
RSS Skole Nyheter Software Klatring Torrentsites Div Dev SMS
Map
Satellite
Hybrid
Oslo
Drammen
E18
Karlstad
Skien
Fredrikstad
Tjøme
Vesterøy
Tjärnö
Skåtøy
Barmen
Brom
Kallandsö
Tromøy
Ramsvik
Uddevalla
Trollhättan
E6
POWERED BY Google
Imagery ©2006 TerraMetrics, Map data ©2006 TeleAtlas - Terms of Use
Submit Query
Done

Where R U? XML-RPC client

This web-based XML-RPC client is a part of Christer Edvartsen's Where R U? project at the Institute of Informatics at Østfold University College in Norway. The project is a part of the Location Aware Systems course.

Using this client you can test the Where R U? XML-RPC server. This client will simulate an XML-RPC message sent from the Where R U? client application that will run on mobile phones using the Series 60 platform. ~~The funcionality is limited to norway, since the tool used for converting lon / lat to UTM only supports the UTM33 zone.~~

If you choose MMS as format, you will need to enter the number of the phone that is supposed to receive the MMS. If you choose mail as format, you need to enter a valid mailaddress that is supposed to receive the mail. If the mailaddress is not valid, the XML-RPC server will not recognize the format.

Updates:

- **2006-05-09: Updated to version 2 of the Google Maps API, since Google have included some more data about Norway! \o/**
- **2006-05-08: The client now uses a method of calculating distances from longitude / latitude, so it does not use UTM33 anymore, and is therefore no longer limited to Norway only.**

Fields marked with * is required.

Please click on the map to set the position of the client.

Format:	MMS [✓ Mail]
Phonenumber:	
Mail:	*
Lon:	*
Lat:	*

If you experience problems with this client, please contact Christer Edvartsen.

As we can see the client consist of some input fields, and a map to use when deciding on the "position" of the client. The mobile client will use a GPS for positioning, but that is not possible in the web-based client, so a map is used to position the client.

The map is generated using Google Maps API which lets you embed Google Maps on your own webpages using only JavaScript [19]. Figure 4.9 shows the code that is needed to display a map on a web page.

Figure 4.9: Showing a map on a web page using Google Maps

```
<script src="http://maps.google.com/maps?file=api&v=2&key=<personal␣key>"
type="text/javascript"></script>
<div id="map" style="width:␣500px;␣height:␣400px;␣align:␣center;"></div>
<script type="text/javascript" language="javascript">
  //<![CDATA[
  var map = new GMap2(document.getElementById('map'));
  map.setCenter(new GLatLng(59.2188, 10.9382), 10);
  map.addControl(new GLargeMapControl());
  map.addControl(new GMapTypeControl());
  map.setMapType(G_HYBRID_TYPE);
  //]]>
</script>
```

The first line includes the JavaScript code needed from google. To gain access to this one needs to sign up for a free unique key at the Google Maps API site. One key is only valid for a single "directory" on a web server, so if the Where R U web-based client was moved to another server (and thereby changes the URL to the service) a new key would have to be generated.

The second line is the container of the map. It is just a regular HTML div tag that has "map" as an id. This id will be used in the JavaScript code later on to tell the API which element it should place the map in. The width and height attributes can be changed to make the map smaller or bigger.

The last chunk of JavaScript code is the part that sets up the map and displays it in the container. First a map object is created, and then it is attached to the map container. Then we center and zoom the map to a specific place (Fredrikstad, Norway in this case) and start at zoom level 10. Last we add some controls to enable users to zoom in and out, and change the map type.

One can also set up custom functions that will be executed when different events happen. Two of the input fields that we can see in the screenshot of the client are Longitude and Latitude. They are there so the user can position the client to the values that are entered in those two fields. A function was then set up to listen for "click" events, and that function will be used when the user clicks somewhere on the map. Figure 4.10 show how this function is set up.

First, the actual function that is used when someone clicks on the map is defined. It is called getp, and it has two arguments. The overlay argument is used if the click was on an overlay on the

Figure 4.10: Adding a listener to an event in Google Maps API

```
function getp(overlay, point){
  document.getElementById('lon').value = point.x;
  document.getElementById('lat').value = point.y;

  map.clearOverlays();
  map.addOverlay(new GMarker(point));
}

GEvent.addListener(map, 'click', getp);
```

map. This is not interesting in this case, so that argument is not used at all. The second argument is a GLatLng object that holds information on where on the map the user clicked. The function will then copy the position of the point to the two input fields, remove previous overlays, and position a new overlay to the position held in the point argument. After the function is a line of code that actually attaches the function to the click event. The addListener function takes three arguments. First is the map object, then a string containing the event we want to add the listener to, and last comes he name of the function that will be used. There are several other events as well, but none of them are needed in the client.

After the user have chosen a position, and filled out the rest of the fields, the information will then be posted and validated. If the information entered is sufficient, an XML-RPC request will be sent to the server. The request will be sent using the phpxmlrpc library as seen in figure 4.11.

Figure 4.11: Sending an XML-RPC request using phpxmlrpc

```
$client = new xmlrpc_client('http://whereru.starzinger.net/xmlrpc.php');

$params = array(
  new xmlrpcval($form['lon'], 'double'),
  new xmlrpcval($form['lat'], 'double'),
  new xmlrpcval($form['format'], 'string'),
  new xmlrpcval($recipient, 'string'),
);

$message = new xmlrpcmsg('find_me', array(new xmlrpcval($params, 'array')));
$r = $client->send($message);

if(!$r->faultCode()){
  // everything went ok
} else{
  // an error occurred
}
```

The first thing that is done is to make a client object. This is done using the xmlrpc_client class. The constructor takes the URL of the server as an argument. There are other ways this can be done as stated in the documentation for phpxmlrpc [17], but this was the simplest. Then we build up an array of arguments that will be sent to the XML-RPC server. Each element of the array must be a valid XML-RPC value, and that is done using the xmlrpcval class that comes with the phpxmlrpc library. The longitude and latitude values will be sent as doubles, and the rest will be sent as strings.

After the arguments have been set up, we will build the complete message. This is done using the xmlrpcmsg class. The constructor of that class takes two arguments. The first is the name of the method on the server that will be executed, and the second argument is that methods arguments. As we saw in the previous section, the name of the method we want here is "find_me", and that method takes an array as an argument. The arguments to the method must be contained within another array.

The last step is to actually send the message. This is done using the send method of the client object. The message that is to be sent must be an xmlrpcmsg, and since we have already made a message, we will just send that one. The return value from the server will in this case be stored in a variable called $r. One can then check the content of $r to see if the request was successful.

The test client was made to test the server. It was also used as reference when the mobile client was made since these two do almost the same things.

4.3 Where R U? SMS specification

The client that will be discussed in section 4.4 will listen on the inbox on mobile phones for incoming messages. When it finds messages what contains a special text pattern, it will recognize these as Where R U? SMS messages. This section will explain the specification of the SMS messages that will be recognized as Where R U? messages by the client.

Since the project is called Where R U?, the client will search for messages that starts with "whereru". After this keyword, information concerning how the user wants the information will follow. The first part contains the medium, and the second part contains the address. An example of an SMS with the Where R U? format can be seen in figure 4.12.

Figure 4.12: Example of a Where R U SMS message that answers using mail

```
whereru mail christer.edvartsen@hiof.no
```

As we can see the SMS starts with the "whereru" keyword followed by "mail christer.edvartsen@hiof.no". When the client finds this message it understands that the sender wants an answer sent to the christer.edvartsen@hiof.no mail address. Early in the project the format was supposed to be even smaller, but this proved to be harder to implement. For instance an SMS containing “whereru christer.edvartsen@hiof.no” could easily be understood by the client. The answer would then be sent to the christer.edvartsen@hiof.no mail address. If the client received an SMS containing “whereru mms” it would not always understand where to send the answer. The reason for this is limitations in the Inbox module used in the client. Information regarding that module can be found in section 4.4. It might also be easier to include other mediums later when the medium is explained using its own keyword in the message.

4.4 Where R U? mobile client

The client is written in Python using Python for Series 60. When writing such applications, one can use many of the standard python library modules along with some made for the Nokia S60 platform. Some of the standard libraries included in the Python for Series 60 distribution [15] are:

- urllib
- socket
- os

The above libraries were all used in the Where R U? client application. [15] states that many of the excluded modules also work in the S60 Python environment without any modifications. An example of this is the xmlrpclib module used in this project. xmlrpclib was installed on the phone and imported into the client application without any modification.

4.4.1 Setting up the application

When making a new application for the S60 platform, some basic configuration needs to be done. Issues such as application name, the size of the window that the application will use, and the default menu. This can easily be done through the appuifw module that comes with the S60 platform. The appuifw module gives the developer an interface to S60 UI application framework. How to set the application title, screen size, default menu and the initial content of the application window can be seen in figure 4.13.

appuifw.app is an instance of the Application class that always exist if the appuifw module is present [15]. title, screen, menu, body and exit_key_handler are some of the attributes of this class.

Figure 4.13: Setting up a S60 Python application

```
appuifw.app.title = u"Where R U?"
appuifw.app.screen = "normal"
appuifw.app.menu = [(u"Start daemon", start_daemon), (u"About", about)]
appuifw.app.body = None
appuifw.app.exit_key_handler = quit
```

“title” is the title of the application that can be seen in the top of the screen.

“screen” is the size of the screen that the application will use. It can be one of: “normal”, “large” or “full”. When using normal as we do in this application, the title pane and softkeys will be visible. If we had used a larger screen, the title of the application would not be visible.

“menu” is a list of menu items. The list in figure 4.13 contains two pairs. The first element of each pair is the name that is visible to the user, and the second element is the name of a callable object that will be executed when the user chooses an element from the menu. In this case, the start_daemon function would be executed if the user had chosen the “Start daemon” option. This menu can be changed from the application at any time, meaning that when a user chooses the “Start daemon” option, it will be replaced by another pair that reads “Stop daemon” and executes the stop_daemon function instead.

“body” can be a UI control that will be visible in the applications main window. In this case the application does not need anything in the main window, therefore it is set to None.

“exit_key_handler” is the last attribute used in the Application class. If this attribute is set, the value is the name of a callable object that will be executed when the user clicks the Exit softkey.

Now that we have set up our application we must include some more logic. The next section will show how to make an application listen to the inbox of the mobile phone for incoming messages. Only SMS messages are supported at the time of this writing [15].

4.4.2 Listen for incoming SMS messages

When this project started in January 2006, it was not possible to listen to the inbox for incoming SMS messages using Python for Series 60, but shortly after a new version was released including an inbox module that could do exactly what the Where R U? application needed.

The inbox module offers APIs to the devices inbox. To be able to listen in the inbox, a method of the inbox class called “bind” can be used. The bind method takes a callable object as an argument which will be executed when the mobile phone receives an SMS. When a user chooses the “Start daemon” option from the menu, the bind method will be called using listen_for_sms as an argument.

listen_for_sms is a function that will scan every incoming SMS message. The function will see if a message contains the format that was explained in section 4.3.

The callable object that is used as an argument to the bind method will receive one argument, and that is the id of the message that just arrived. This id can then be used to retrieve the content of the SMS message using the content method available in the inbox class. The content of the message will be returned as a unicode string, and can then be analyzed using regular string methods available from Python. Figure 4.14 shows how to fetch the content of a new message, and see if the content of the message starts with “whereru ”. message_id is the unique id of the SMS message that just arrived in the inbox, and the content of that message is retrieved using the content method from the inbox class. As we can also see from figure 4.14, a local inbox variable called “inb” is made to have an updated version of the inbox. The content is then stored in another variable called “message” and by using the startswith method, we can see if the SMS message starts with the text “whereru ”. If this is the case, we need to handle the message as a Where R U? request.

Figure 4.14: Analyzing the content of an SMS message

```
def listen_for_sms(message_id):
  inb = inbox.Inbox()
  message = inb.content(message_id).lower()

  if message.startswith("whereru␣"):
    # we have a Where R U? SMS. Do some more here
```

4.4.3 Fetch position data from GPS

To gain information on the whereabouts of the mobile phone, a bluetooth GPS receiver has been used. The Where R U? application must access this receiver to fetch position information and post this to the XML-RPC server. A simple class that uses the standard socket module with some S60 extensions was made for this. The class can be found in figure 4.15.

The constructor of the class opens a socket using the standard socket method of the socket module using a constant available on the S60 platform, AF_BT, and a standard constant, SOCK_STREAM. AF_BT is a constant that represents the bluetooth address family, and SOCK_STREAM which represent a socket type. Next we use the bt_discover method to discover the bluetooth devices connected to the phone. When using this method, the user of the phone will see a list of devices on the display of the phone, and must choose one of the devices in the list. The function returns a pair where the first element is the address of the device chosen by the user, and the second is a list of available

Figure 4.15: Class for reading information from a GPS receiver

```
class btReader:
  def __init__(self):
    self.sock = socket.socket(socket.AF_BT, socket.SOCK_STREAM)
    try:
      address, services = socket.bt_discover()
      self.sock.connect((address, services.values()[0]))

      self.f = self.sock.makefile()
    except socket.error:
      appuifw.note(u"Could not connect to device. Aborting...", "error")
      quit()
  def readLine(self):
    line = self.f.readline()
    return line.strip()
```

services on the device. When the user has chosen a device, the constructor connects to it using the standard connect method that takes a pair as an argument, where the first element is the address to connect to, and the second element is the name of the service. The last thing done in the constructor is to make a file object associated with the socket so one can read from the connection in a similar fashion as when reading regular files. If somehow this part fails and raises an exception it will be caught by the application, present an error to the user and quit.

Once the bluetooth device is connected to the socket, a method called readLine can be used to read from it. Since we already have a file object associated with the socket, we can simply use that objects readline method. Each line returned from the GPS receiver will be according to the NMEA 0183 standard. NMEA 0183 is a data specification for communication between marine electronics and GPS receivers. An NMEA line can be seen in figure 4.16, and the description of the different parts of the line is found in table 4.2.

Figure 4.16: NMEA 0183 line

```
$GPRMC,141258.524,A,5922.0923,N,01093.6868,E,0.00,11.23,010306,,,*36
```

We are interested in lines starting with $GPRMC since they concern the position. The GPS receiver returns other information as well, but the $GPRMC lines are the only ones used in this project. The status of the line is important as well since it informs us if the information is valid or not. The application will then read lines from the GPS until it finds a valid line.

The longitude and latitude values are specified in degrees and minutes and will have to be converted to degrees and seconds so we can store the values as doubles. How they are converted

Table 4.2: Description of an NMEA line

Field	**Example**	**Comment**
Sentence ID	$GPRMC	
UTC Time	141258.524	hhmmss.sss
Status	A	A = Ok, V = Warning
Latitude	5922.0923	ddmm.mmmm
N/S Indicator	N	N = North, S = South
Longitude	01093.6868	dddmm.mmmm
E/W Indicator	E	E = East, W = West
Speed over ground	0.00	Knots
Course over ground	11.23	Degrees
UTC Date	010306	DDMMYY
Magnetic variation		Degrees
Magnetic variation		E = East, W = West
Checksum	*36	

can be seen in figure 4.17. When the position of the mobile phone has been found, we can send this information to the XML-RPC server. How this is done is explained in the next section.

Figure 4.17: Converting longitude and latitude to degrees and seconds

```
line = '$GPRMC,141258.524,A,5922.0923,N,01093.6868,E,0.0,11.23,010306,,,*36'
data = line.split(',')
lon = data[5]
lat = data[3]
lon = str(float(lon[:3]) + (float(lon[3:]) / 60))
lat = str(float(lat[:2]) + (float(lat[2:]) / 60))
```

4.4.4 Send an XML-RPC request

The xmlrpclib library has been used to take care of the XML-RPC part of the client application. It works the same way as the client part of the phpxmlrpc library does for PHP. It simply translates Python objects such as lists and strings, to the XML counterpart required by the XML-RPC server.

To be able to use the methods available from the server, we will need to connect to it first. This can be done by using the ServerProxy method from the xmlrpclib class available in xmlrpclib. How this is done in the Where R U? client can be seen in figure 4.18.

The ServerProxy method returns a serverproxy object that have the same methods as the XML-RPC server has. In this case “find_me” and “fetch_map”. These methods can then be run by using

Figure 4.18: Connecting to the XML-RPC server using xmlrpclib

```
xmlrpc_server_url = "http://whereru.starzinger.net/xmlrpc.php"
xmlrpc_server = xmlrpclib.ServerProxy(xmlrpc_server_url)
```

the object returned from the ServerProxy call. As we can see from figure 4.18, the serverproxy object is stored in the xmlrpc_server variable.

The server has implemented two methods: One is used when the server is to handle everything, and one if the server is only to fetch the map, and let the client do the rest. Which method to use depends on the medium specified in the SMS messages that the client receives. We will now look at how to let the server do all the work, and then we will see how to retrieve an URL to a map generated by the server so the client can send the map to another user via MMS.

Figure 4.19 shows how to execute the find_me method on the server. As we can see, the methods implemented on the server has become methods available in the serverproxy object stored in the xmlrpc_server variable. The argument to the method is sent as a list, and is converted by the xmlrpclib library to the appropriate structure needed by the method on the server. Since the method on the server expects the longitude ant latitude variables to be doubles, they need to be converted to a python floating point object using the float function before they are sent to the server. As we have seen, the find_me method on the server returns a boolean that is true if the request was successfully handles, and false otherwise. The output from the method is stored in the result variable that can be used for error checking later on.

Since the XML-RPC server can not handle MMS requests, the client application much check the contents of the medium variable before it executes a method on the server. The next section will describe how to use the unofficial MMS module to send MMS messages from the client application.

Figure 4.19: Execute a method on a remote XML-RPC server

```
lon = float(gps_packet[0])
lat = float(gps_packet[1])
result = xmlrpc_server.find_me([lon, lat, medium, recipient])
```

4.4.5 MMS handling

Since Python for Series 60 has not yet implemented support for sending MMS message, the Where R U? client application has used an unofficial module. The module is simple to use and requires very little code to send an MMS. Figure 4.20 shows the code that is needed. recipient is the mobile number that will receive the MMS and the subject of the message is specified in the subject variable. The name of the image we want to send is stored in the image variable. All variables must be unicode strings [8].

Figure 4.20: Sending an MMS using Python for Series 60

```
retcode = mmsmodule.mms_send(recipient, subject, image)
```

The image must first be downloaded from the internet before we can send it from the client. The fetch_image method on the server is used for this. The method is executed and an URL to a map image is returned to the client. To download the image, the standard Python urllib module is used.

That sums up how the different parts of the Where R U? project has been implemented. Not all elements of the different applications have been included, but you should have a good overview on how the main parts work. Some possible extensions is listed in chapter 7.

4.5 Summary

As we now have seen we have different applications written in different programming languages. The XML-RPC technology enables these parts to communicate with each other using XML and the HTTP protocol. The key part here is the XML-RPC libraries that has been made for these languages. When developing a project such as this one will most probably need to make use of existing libraries instead of making everything from scratch.

Having read this chapter, you should now posess the information needed to maybe continue developing the Where R U? project, or start a similar project. The next chapter goes into detail on the different goals set up in the project along with the results.

Chapter 5

Results

This project has been developed in small stages. Most of the technology used in the Where R U? project was new to me when starting it, and because of this it was very hard to set timelimits for the main goals.

This chapter will sum up the results made during the course of the project. Several goals have been changed due to lack of technology, or new technology appearing. Each section in this chapter will describe main goals in the project.

5.1 Development environment

The first goal was to simply make a computer ready for developing software for the Nokia S60 platform. This was done by installing the software available at Nokias developer site [20]. The software needed for this project is free and easy to install.

5.2 Hello World

The second goal was to have a simple application running on the Nokia 6680. Most projects start off with a simple Hello World application, and so did the Where R U? project. After the Hello World application was completed, more advanced goals could be set that were relevant to the project.

5.3 Connect to GPS

The next goal after getting the Hello World application up and running was to connect the Nokia 6680 to a GPS receiver and read information from it. The type of the receiver is not that important since most receivers deliver position data in a standard way called NMEA. This means that users who want to test Where R U? do not need to have the same GPS receiver that is used for developing and testing. This goes for the mobile phone as well. The requirement is that the mobile phone is running the Nokia S60 platform.

5.4 Sending MMS from Python for Series 60

This topic has been changed a couple of times during the project. In the beginning of the project, the mobile phone was supposed to send the MMS itself. As mentioned above, this was later changed due to Python for Series 60 not having an official MMS module. After this was discovered, the MMS handling was moved to the XML-RPC server instead. Later in the project an unofficial MMS module was released for Python for Series 60, so the MMS handling was again moved to the mobile client. At the time of this writing, MMS is not implemented in the XML-RPC server at all, but if an MMS API were to be made available, it should not be too much work to implement it on the server instead of the mobile phone. This might be a good idea since the mobile phone is less powerful than the XML-RPC server and has less bandwidth.

5.5 XML-RPC server

After some of the goals concerning the start of the mobile client had been met, the XML-RPC server was started. I had some prior knowledge of the phpxmlrpc library, so this part of the project did not have as small goals as the other parts.

First, only one method was implemented. This method was supposed to do all the work, including sending mail and MMS messages.

When the unofficial MMS module for Python for Series 60 was released, another method was to let the mobile phone fetch the map and send the MMS itself. This was done at a later stage since the MMS module appeared close to the end of the project.

As mentioned in chapter 3, the server first used UTM when generating the bounding box that the WMS server needs. When the proof of concept was ready, the server still used UTM. This part was changed late in the project as an enhancement of the server.

5.6 XML-RPC test client

During the development of the XML-RPC server, a web based test client was made to make sure that the server part worked as intended. The test client makes use of Google Maps to simulate the positioning of the client, and can request both MMS and mail. The test client was made at the time when the server was supposed to handle MMS. The requesting of MMS was not removed from the client after the MMS module appeared since MMS support might be added to the server in the future.

5.7 Where R U? SMS specification

The SMS format was specified before the mobile client was to be finished. The specification had to be changed later due to a shortcoming in the inbox module used in the mobile client that has been discussed earlier in this report.

5.8 Mobile client application

When the XML-RPC server was completed and tested using the test client and the SMS format was specified, development of the mobile client was started. Since all the previous goals related to Python for Series 60 were to be used in this application, they were simply combined into one application. Key features like listening to the inbox, and sending XML-RPC requests were still missing at this point, and were implemented after the previous goals were combined.

At this point of the development of the client, listening to the inbox could easily be done using the inbox module for Python for Series 60. This module became available during this project, and could not have been implemented at the early stages. One minor shortcoming of this module made a change to the Where R U? SMS format, as was mentioned earlier in this report. The shortcoming relates to fetching the phonenumber of the person sending the SMS message. If the person sending the message is located in the contact list of the phone, you can not be certain that you get the correct phone number. Therefore the SMS format needed to include a recipient part along with the medium to receive the answer.

When figuring out how to listen on incoming messages the only part missing to have a proof of concept was to send XML-RPC requests from the mobile client. This was done using the Python xmlrpclib library which provides XML-RPC client access. I had used this library before doing this project, so it did not take long until the proof of concept was done. At first the only medium that worked was mail since the MMS module had yet to be released.

This sums up the different goals that was set and met during the project. The next chapter contains some of the tests made concerning the software produced in this project.

Chapter 6

Testing

This result will deal with the different tests that was made with the software produced in this project. The XML-RPC server and the mobile client have been extensively tested. The web-based client was only made to test the XML-RPC server, to that part has not been tested thorough itself.

6.1 XML-RPC server

As mentioned earlier, when the XML-RPC server was almost finished, a web-based client was made to test it. Debugging in the form of logging was added to the XML-RPC server do I could easily see what happened when the client made a request. The way this was done was to add calls to a function that wrote a string to a log file where it was needed. The log file contains information about the request such as longitude, latitude, medium and so forth. A section of the log file that contains information about one request can be seen in figure 6.1.

The debugging will show if some of the functions fail to do what they are supposed to. If for instance the function that downloads the map from the WMS for some reason does not save the image on the server the log will show this. By looking for errors in the log file one can find functions that do not work and start investigating these to find possible bugs.

The XML-RPC server is written in PHP and has only been tested on the Linux platform. It will most likely run on the Windows platform as well without the need to modify the source code.

Figure 6.1: Debugging the XML-RPC server

```
Thu Jun  1 14:00:16 2006 (1149163216)
------------------------------------------------------------
Method: find_me
Longitude: 10.940322876
Latitude: 59.2145208541
Medium: mail
Recipient: christer.edvartsen@hiof.no
Image type: png
Width: 800
Height: 600
Map URL: http://basiswms.statkart.no/servlet/com.esri.wms.Esrimap?WMTVER=1.0&
REQUEST=map&SRS=EPSG:4326&FORMAT=PNG&BGCOLOR=0x23f3f5&TRANSPARENT=TRUE&STYLES
=default,,,,,,,,,,,,,,,&LAYERS=Landtone,Vann,Markslag,Vannkontur,Elv,Jernbane
,Kommunegrenser,Hoydekurver,Bygninger,Bygninger_grunnriss,N50Bebyggelse,
Bebyggelse,Adresser,Gatenavn,Stedsnavn,Fylkesnavn,Veg&ServiceName=
Topografisk_Norgeskart_wms&BBOX
=10.9154617848,59.2017939502,10.9651932381,59.227247758&WIDTH=800&HEIGHT=600
Mail sent.
============================================================
```

6.2 Mobile client application

The same debugging technique was used on the mobile client application. A flag is set to either true or false informing the application to write debug messages to a file or not. This file can then be retrieved later and will show information about all requests made. Is a request failed it will also be seen in the file. Since the application is supposed to be running in the background it is not a good idea to use the regular information pop ups available from the appuifw module in Python for Series 60. The user that is running the Where R U? application should not need to look at the application window for error messages.

As stated earlier in the report the mobile application has been tested on the Nokia 6680 mobile phone using the Holux Bluetooth GPS receiver. In theory the mobile client application will run on any mobile phone sporting the Nokia S60 platform with Python for Series 60 and the required modules installed.

The next chapter contains topics that are up for discussion, possible extensions to the project, and a final conclusion.

Chapter 7

Discussion, Future Work and Conclusions

This chapter will provide discussions about other possible ways the project could have been done, some ideas for future work, and finally a conclusion.

7.1 Discussion

7.1.1 Other programming languages

As stated before there are other programming languages developers can use to make applications for mobile phones using the Nokia S60 platform. Developers can program in C++ using the native Symbian OS API, or with the Java language using MIDP 1.0 or MIDP 2.0 with a range of additional Java Specification Requests (JSRs).

I had some experience with C++, but after consulting the Symbian OS APIs, it was obvious that it would be too cumbersome to do this project using the C++ language. One can achieve much more control over the mobile phone using C++, but the timeframe of the project would not be long enough.

The other possibility was to use the Java programming language, and I did not have to much experience with that language, so that was out of the question. One might also achieve more control using Java, but then again, the time it would take to learn the language would be too long. The Location Aware Systems course that this project is a part of is not really about learning new programming languages, so using Python seemed like the best solution for this project, since this was the programming language I had most experience with.

7.1.2 Personal rights

When developing an application like Where R U? one also needs to take a look at the personal rights aspect. A system where person A can gain information on where person B is located without person B knowing about it might not be too popular amongst all people. This, however, shows how advanced the devices we walk around with every day have become. Most people might be afraid of others gaining access to their personal computers, but they might not think the same of their mobile phones.

7.2 Future work

The Where R U? project can easily be expanded to include some more interesting features. Some possible expansions to the project will be listed in the next sections.

7.2.1 Security

The mobile client should have the possibility to choose which users are to be allowed to send Where R U? requests. This might be as simple as specifying a list of allowed numbers. One problem with this might be the functions available in the inbox module. As stated earlier in the report, if a person sending an SMS is on the contacts list on the phone, we can not be sure if we get the correct phone number using the available functions in the inbox module. One possible solution to this problem would be to extend the inbox module for Python for series 60 by adding a function that always returns the number of the sender, regardless if the user sending the SMS is in the contact list or not. This would require some knowledge of C++ and some knowledge of how the Symbian OS APIs work.

7.2.2 Logging

Once the Where R U? mobile client is started, it should produce a log that the user could retrieve later to see who has been using the service. Another way to do this could be to let the XML-RPC server enable logging, and somehow give the user access to his/her logs on the Internet.

7.2.3 More user options

As of now, the users of the Where R U? project does not have many options. The only option is to decide on the medium that the mobile client will answer to. This is done via the SMS sent to the mobile client. The following sections list some possible options that could be implemented.

Choose WMS via SMS message

The XML-RPC server now use the Statens Kartverks WMS server that only includes data about Norway. This means that the Where R U? project can not be used outside of Norway without some changes. An option could be that the user sending the SMS could specify the WMS to be used in the SMS message. This way the users could use specific WMS servers if they want some special information on the map.

Choose layers

To let the user decide on which layers to be put on the map could also be useful. This would require some knowledge of the WMS server used, but combined with the idea from the previous section, this could be a useful option. This might also be specified in the mobile client so that the user running the client could specify some default options.

Size of map image

The size of the image is now hardcoded into the mobile client. The default for mail and MMS requests is 800 x 600 pixels and 640 x 480 pixels respectively. The user running the client should be able to specify other defaults, and the user sending the Where R U? SMS should also be able to specify the size of the image via the SMS message. The XML-RPC server is capable of this already, so the only part that needs to be changed is the mobile client, and possibly the format of the SMS message if the user should be able to specify sizes in the message.

As stated earlier in this report, the map is generated by a WMS based on a bounding box and some other options. The XML-RPC server has hardcoded the radius of the box to 2000 meters. This is also an option that could be specified in the mobile client, and in the SMS messages sent to the client.

7.2.4 Support for more mediums

Only mail and MMS have been implemented for the project so far. There are many other mediums that could be interesting to include in a project such as this, and some of them are:

- RSS (Submit the request to an RSS feed available on the Internet)
- KML (Receive the answer in the form of a KML file that can be imported into Google Earth)
- Blog (Submit the request to an online Blog)

7.3 Conclusions

After doing a project such as this I have realized the importance of technologies such as Nokia S60 and above all Python for Series 60. With this platform users can make useful applications with little effort. I have also realized how cutting edge this research is when modules used in the Where R U? project were released during the project. The technology is fairly new, and some shortcomings are still to be fixed. However, the developers at Nokia are active in the community and are very helpful to other people that want to contribute with their own modules, such as the MMS module used in this project.

One negative aspect of developing software using Python for Series 60, is that the user who wants to install the software on the phone might need to install several other pieces of software first, as opposed to creating a midlet using the Java programming language. To be able to use the Where R U? mobile client one needs to first install the Python for Series 60 platform, then the libraries that are not part of the distribution, like xmlrpclib, and finally one can install the application itself.

This project has also brought to my attention the personal rights aspect when developing something like this. Using the modules that Nokia provides it is very easy to eavesdrop on an inbox by installing a small piece of software on the phone. Most people might not be aware of this and therefore do not treat their mobile phone with as much consideration as say a personal computer.

All in all this project has been quite interesting, and it is very nice to see a growing and helpful community were professionals from Nokia and other users interact to help each other make useful and exciting software. The project met its goals and is an good example of a proof of concept.

References

[1] Wikipedia, “Smartphone,” Mar 2006. [Online]. Available: http://en.wikipedia.org/wiki/smartphones

[2] S. Kartverk, “Statens kartverk,” May 2006. [Online]. Available: http://www.statkart.no/IPS/

[3] O. G. C. Inc., *Web Map Service*, Open Geospatial Consortium Inc., August 2004.

[4] Wikipedia, “Nokia series 60,” Mar 2006. [Online]. Available: http://en.wikipedia.org/wiki/S60

[5] Nokia, “S60 platform,” Mar 2006. [Online]. Available: http://www.forum.nokia.com/main/0,6566,010_400,00.html

[6] Wikipedia, “Global positioning system,” May 2006. [Online]. Available: http://en.wikipedia.org/wiki/GPS

[7] P. S. Foundation, “The python programming language,” May 2006. [Online]. Available: http://www.python.org/

[8] K. Kujansuu, “Mms module for python for series 60,” May 2006. [Online]. Available: http://nysse.lapanen.org/python/MMS/

[9] eriksmartt, “Developer discussion boards - python,” Feb 2006. [Online]. Available: http://discussion.forum.nokia.com/forum/showthread.php?t=74940

[10] T. P. Group, “Php: Hypertext preprocessor,” May 2006. [Online]. Available: http://www.php.net/

[11] X.-R. for PHP developers, “Xml-rpc for php,” May 2006. [Online]. Available: http://phpxmlrpc.sourceforge.net/

[12] ——, “Xml-rpc for php,” May 2006. [Online]. Available: http://docs.python.org/lib/module-xmlrpclib.html

[13] M. Lutz and D. Ascher, *Learning Python, Second Edition*. Sebastopol, CA: O’Reilly Media, Inc., 2003.

[14] M. Lutz, *Programming Python, Second Edition*. Sebastopol, CA: O’Reilly Media, Inc., 2001.

[15] Nokia, *PyS60 Library Reference, Release 1.3.1 final*, Nokia Corporation, January 2006.

[16] M. Lindh, “Onemap coordinate converter,” Mar 2006. [Online]. Available: http://onemap.org/proj.php

[17] E. Dumbill, G. Giunta, *et al.*, “Xml-rpc for php manual,” May 2006. [Online]. Available: http://phpxmlrpc.sourceforge.net/doc-2/

[18] I. Boutell.Com, “Gd graphics library,” May 2006. [Online]. Available: http://www.boutell.com/gd/

[19] Google, “Google maps api version 2 documentation,” May 2006. [Online]. Available: http://www.google.com/apis/maps/documentation/

[20] Nokia, “Forum nokia,” Mar 2006. [Online]. Available: http://www.forum.nokia.com/

[21] J. Brewer, “Extreme programming faq,” May 2006. [Online]. Available: http://www.jera.com/techinfo/xpfaq.html

List of Figures

List of Tables

Appendix A

Technical terms and abbreviations

API

Application Programming Interface.

Blog

Weblog.

Bluetooth

An industrial specification for wireless personal area networks.

C++

A general-purpose computer programming language.

Extreme programming (XP)

A software engineering methodology for the

GPS

Global Positioning System.

J2ME

Java Platform, Micro Edition.

Java

An object-oriented programming language.

JSR

Java Specification Request.

KML
: Keyhole Markup Language

MIDP
: Mobile Information Device Profile.

MMS
: Multimedia Messaging Service.

NMEA
: National Marine Electronics Association.

OS
: Operating System.

PDA
: Personal Digital Assistant.

PIM
: Personal Information Manager.

pys60
: Python for Series 60.

RSS
: Really Simple Syndication.

S60
: Series 60.

SMS
: Short message service.

UI
: User Interface.

URL
: Uniform Resource Locator.

UTM
: Universal Transverse Mercator.

W3C
: World Wide Web Consortium.

WAP
: Wireless Application Protocol.

Web service
: W3C defines a Web service as a software system designed to support interoperable machine-to-machine interaction over a network.

XML-RPC
: XML-RPC is a remote procedure call protocol which uses XML to encode its calls and HTTP as a transport mechanism. development of software projects.

Appendix B

Twelve practices of the Extreme Programming methodology

The twelve practices of Extreme Programming as listed on [21] is a follows:

1. **The Planning Game:** Business and development cooperate to produce the maximum business value as rapidly as possible. The planning game happens at various scales, but the basic rules are always the same:

   (a) Business comes up with a list of desired features for the system. Each feature is written out as a User Story, which gives the feature a name, and describes in broad strokes what is required. User stories are typically written on 4x6 cards.

   (b) Development estimates how much effort each story will take, and how much effort the team can produce in a given time interval (the iteration).

   (c) Business then decides which stories to implement in what order, as well as when and how often to produce a production releases of the system.

2. **Small Releases:** Start with the smallest useful feature set. Release early and often, adding a few features each time.

3. **System Metaphor:** Each project has an organizing metaphor, which provides an easy to remember naming convention.

4. **Simple Design:** Always use the simplest possible design that gets the job done. The requirements will change tomorrow, so only do what's needed to meet today's requirements.

5. **Continuous Testing:** Before programmers add a feature, they write a test for it. When the suite runs, the job is done. Tests in XP come in two basic flavors.
   (a) Unit Tests are automated tests written by the developers to test functionality as they write it. Each unit test typically tests only a single class, or a small cluster of classes. Unit tests are typically written using a unit testing framework, such as JUnit.
   (b) Acceptance Tests (also known as Functional Tests) are specified by the customer to test that the overall system is functioning as specified. Acceptance tests typically test the entire system, or some large chunk of it. When all the acceptance tests pass for a given user story, that story is considered complete. At the very least, an acceptance test could consist of a script of user interface actions and expected results that a human can run. Ideally acceptance tests should be automated, either using the unit testing framework, or a separate acceptance testing framework.
6. **Refactoring:** Refactor out any duplicate code generated in a coding session. You can do this with confidence that you didn't break anything because you have the tests.
7. **Pair Programming:** All production code is written by two programmers sitting at one machine. Essentially, all code is reviewed as it is written.
8. **Collective Code Ownership:** No single person ”owns” a module. Any developer is expect to be able to work on any part of the codebase at any time.
9. **Continuous Integration:** All changes are integrated into the codebase at least daily. The tests have to run 100% both before and after integration.
10. **40-Hour Work Week:** Programmers go home on time. In crunch mode, up to one week of overtime is allowed. But multiple consecutive weeks of overtime are treated as a sign that something is very wrong with the process.
11. **On-site Customer:** Development team has continuous access to a real live customer, that is, someone who will actually be using the system. For commercial software with lots of customers, a customer proxy (usually the product manager) is used instead.
12. **Coding Standards:** Everyone codes to the same standards. Ideally, you shouldn't be able to tell by looking at it who on the team has touched a specific piece of code.