A Location Bound Media Client for Sony Ericsson P800/P900

Digital Maps Project

Arne Enger Hansen

A Location Bound Media Client for Sony Ericsson P800/P900: Digital Maps Project

Arne Enger Hansen

Table of Contents

List of Figures

Preface

Abstract

This projects aims to implement a framework for creating, delivering and storing location bound media such as images, audio and video. The main focus of this report is creating and delivering such media. A client for creating location bound media has been implemented for the Sony Ericcson P800/P900. For positioning we use a Bluetooth GPS device from Socket Communications Inc. MMS is used as a content delivery protocol. This application is developed using open protocols and tools.

Prerequisites. It is assumed that the reader is familiar with common mobile phone terminology. Some knowledge about programming, XML, Java and the Java Virtual Machine is also assumed.

Outline. Outline

Project OneMap. The work presented in this report is the main deliverable in the Digital Maps course 2004[5] [43] which is part of the Environmental Computing specialization at the master program at Faculty of Computer Sciences, Østfold University College, Halden, Norway[4] [43]. Digital Maps are considered as a sub project of Project OneMap[3] [43], in the sense that selected results from the student projects will be used in the OneMap infrastructure. The students are considered to be part of the OneMap team in the course period. Project OneMap is a long term effort contributing to the fusion of standard web technologies and geographic content, often referred to as the GeoWeb.

Chapter 1. Introduction

In this chapter the project is introduced. It is useful for readers to understand why and in what context this project has been done. Other related projects are introduced, which helps seeing this project in a bigger picture. Location Based Services are explained, as the project can be described as a Location Based Service, and is an important motivation behind this project.

1.1. Location Based Services

Location Based Services (LBS) is said to become one of the most important mobile services in the years to come [20] [44]. This term describes applications whose behavior/output is determined by location. Location awareness can be obtained using a GPS device, or in the case of mobile phones using the cell info from the network. Another, but more cumbersome, approach is simply to let the user enter a location manually, by clicking on a map or by entering a city, region or actual coordinates. The various approaches offer different levels of accuracy, and is suitable for different applications.

Some location based services already exists today, mostly based on the most primitive approach. E.g. one can ask (by SMS) to get a list of recommended restaurants in a city. This is a sort of location aware yellow pages. Other services that are likely to surface as the more sophisticated methods for positioning becomes more widely available are: current area map, directions from current location to another (eg. a recommended restaurant), location bound media (such as pictures, text, audio, video) or messages (possibly containing media), position tracking, find-a-friend (a new one close by or find the location of an existing one!) and location aware traffic alerts just to mention a few.

Devices suitable for LBS. Mobile phones and PDAs are particularly interesting in this context as they are the most portable. Phones also have the advantage of being connected to a network. Several newer phones also include PDA functionality, audio recorders and cameras. Such additional functions will become more and more common.

Laptops and portables are also an somewhat interesting platform, and LBS already exists such as GPS applications used in boats. Several other devices such as cameras, cars, robots (such as fully automated lawn-mowers), RC cars/planes etc. may benefit from positioning and LBS.

It is however likely that several of the mentioned platforms will converge into one single device. Already several phones include PDA functionality and cameras, and this development is likely to increase in the near future. The performance difference between a top end Phone/PDA and laptops is likely to become less significant as the former becomes more powerful.

1.2. Project OneMap

OneMap is a long term project aiming to integrate geographic content with standard web technologies. It is based at Østfold University College, but external recourses are welcome to contribute. The project is all open source, and uses open tools and protocols as far as possible. Geographical data is stored in vector format. There are three main components that make up OneMap: Repository, Gateway and Clearinghouse.

The Repository. The Repository stores the geographic data. Data is stored in GML[22] [44] format, and is accessed trough WFS[23] [44]. The Repository may contain vast amounts of data. As a consequence a distributed storage infrastructure is used. This enables data to be spread on a number of hosts, also duplicated to ensure data is available.

The Gateway. The Gateway is a web interface to the geographic content stored in the Repository. The content is converted to SVG[24] [44] and displayed in a browser. Content can also be accessed as GML or rendered as a

JPEG picture.

The Clearinghouse. The Clearinghouse is an infrastructure for adding geographic content to the Repository. To ensure the quality of new geodata and to integrate it with the existing geodata, the principles of peer review is employed. This facilitates building the map in an incremental manner.

1.3. Digital Maps Project

The project described in this report is part of the Digital Maps[5] [43] course at Østfold University College. This project is just one of several projects surrounding the OneMap Mobile Platform (described in Section 3.1, “OneMap Mobile Platform Architecture” [13]), and has to be seen in context with the others. In particular this project is concerned with implementing the on-phone software part of the P800 plug-in. The server part of the plug-in is implemented by Christer Stenbrenden, and the OneMap Mobile Platform is implemented by Filip Hammerstad. Another plug-in is being developed by Christer Edvartsen and Oddbjørn Kvalsund.

1.3.1. Project Description

> Implement a OneMap Client on a Sony-Ericsson P800 with a Bluetooth connected GPS.
>
> —Gunnar Misund: Digital maps projects[6] [43]

Though somewhat diffuse initially, this project description has evolved during the project. We aim to implement a location based service for the P800. More specifically the purpose of this project is to implement an application for creating and sending location bound media. The term location bound media refers to media, such as images, audio or text, which is tagged with the exact location where it was created.

1.4. Been There - Done That

The working title for our application is "Been There - Done That", which describes in a good way what the application does. The infrastructure will allow location bound media to be sent as a MMS message; either to another phone, or to the a server to be saved as a geographic feature in the OneMap Repository.

1.4.1. Scenarios

Note

This section is coauthored with Christer Stenbrenden.

There are several possible uses for such a service, a few examples will follow.

Field report. An environmentalist discovers a leakage of toxic waste. He goes on to measure the amount of toxins in the ground. For each measurement he records the results on his mobile phone. At the same time he activates his GPS device, and records each position he measures. Back at the office he can use this data to draw up a map of the area affected by the leakage. In combination ¨ with other geographic information he is able to predict how the waste will spread over time, and can plan effectively and correctly what counter actions to take.

Error report. An error is discovered on a railroad track. The train conductor uses the camera on his phone to document the error, and records a voice note describing what he thinks is wrong, and sends the message to the service department. They see the pictures, and hear what the conductor says, but in addition they can see exactly where the error is located. A service team is given driving instructions, and can locate the fault easily. Meanwhile the train company, knowing exactly where the fault is located, can take appropriate action to minimize the effect on the affected routes.

Holiday diary. You have just booked your tickets for this year's vacation and are soon leaving for Greece with

your new Sony Ericsson P900 and it's bluetooth GPS module. While travelling around beautiful Greece you'll find lots of wonderful things to see and hear, so you record sounds, takes pictures and record movies including your own written descriptions (maybe) and you'll send it to your server at home. Your friends and family back home can through your website keep track of where you are, what you are experiencing at all the places you travel. And when you come home, you'll have a great travel diary/log for later rememberance of the fantastic trip.

The Biologist. Tracking animals and their tracks is a tedious buisness. A biologist with a system based on BTDT could easily walk through the huge forrest taking pictures of excrements, following tracks or recording sounds and send it to the server. No more than her phone is needed to build up a database of her favorite animal's daily routines. The webinterface shows the different animals plotted on a map with different markers and clicking on them brings up the recorded data on that particular spot. Selecting data from one particular animal (or plant?) in any period of time is also possible. Each recorded MMS can be further edited at home on the computer and connected to live samples of the findings etc.

1.4.2. Functionality

The application developed lets users create and send location bound media. The application is designed to be easy to use. The user interface will provide methods for recording a message and sending a message. After the user has started recording a message she will switch to other applications to create the media objects she wants to send. These will automatically be added to the message when she later switches back to BTDT and activates the send function. More detail on functionality can be found in Chapter 3, *Design* [13].

1.4.3. Equipment Used

The target platform for the application is the P800 [http://developer.sonyericsson.com/site/global/products/phones/p800/p800.jsp] and P900 [http://developer.sonyericsson.com/site/global/products/phones/p900/p900.jsp] mobile phones (see Figure 1.1 [3]) from Sony Ericsson. For obtaining position we use Bluetooth GPS Reciever [http://www.socketcom.com/product/GP0804-405.asp] (see Figure 1.2 [4]) from Socket Communications Inc. This constitutes the hardware on the client side of the application.

Figure 1.1. The P800(left) and P900(right) phones

Figure 1.2. The Bluetooth GPS Receiver from Socket Communications Inc.

The server uses an Fasttrack M1306B [http://www.wavecom.com/Products_V2/product.php?prs_id=51&prg_id=14] GSM/GPRS modem from Wavecom Inc. to receive and send MMS messages.

Figure 1.3. WaveCom GSM/GPRS Modem

Chapter 2. Background

This chapter describes the technologies related to implementing the application described. This helps readers understand the following chapters, which gives more detail on the design and implementation of the project. The technologies described here are used, eighter directly or indirectly, by the application developed.

2.1. Positioning

Positioning is essential for location based services. There are several ways to obtain a position automatically, and for some applications it's sufficient to let the user enter the position manually.

GPS. GPS[7] [43] is a system for pinpointing the location of a receiver device on earth. The system consists of 24 satellites in orbit, and a receiver. The receiver uses the location of at least 3 satellites, and the distance from the unit to each satellite to deduce it's own position using a method known as trilateration. This positioning method is very accurate, but it may be difficult to get contact with the satellites in certain weather conditions, in cities or indoors. Also this requires additional equipment which is expensive. For this project we use a Bluetooth connected GPS device from Socket Communications Inc.

Cell based positioning. A less expensive method for positioning in an GSM handset is using the GSM network[25] [44]. The handset knows which cell it is connected to, and the coverage of this cell. This can give an idea of the area in which the handset is located (Figure 2.1 [5]). If the handset is able to get a connection to more than one cell a more accurate position can be obtained (Figure 2.2 [6]). By measuring the distance to the transmitters (signal time) an accuracy as low as 50m can be achieved. This method demands no additional hardware and can give fairly accurate results. A product using this method is developed by Cel-Loc Location Technologies Inc.[26] [44]

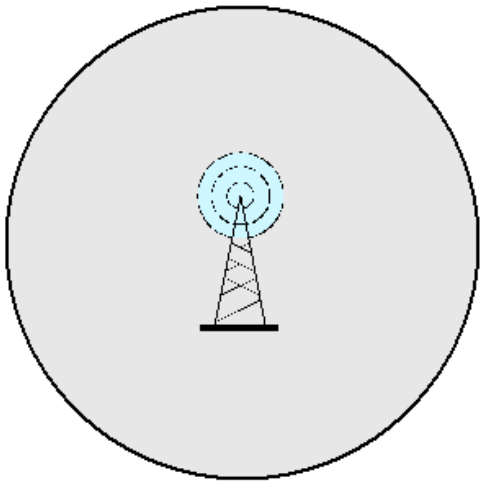

Figure 2.1. Cell based positioning

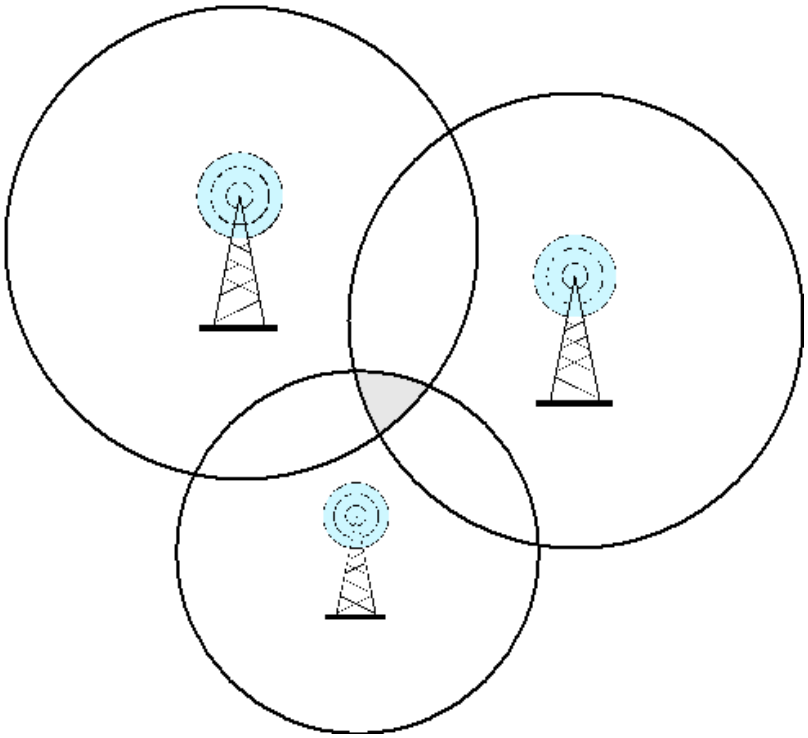

Figure 2.2. Cell based positioning with 3 cells

A-GPS. A-GPS (Assisted GPS) is a combination of GPS- and Cell-based methods. It's an accurate positioning method, and contrary to GPS it can operate without a line of sight to the satellites. A product using this method has been developed by SnapTrack Inc. [27] [44]

Manual positioning. If no other method of positioning is available it is possible to let the user enter the position manually. In some cases it's sufficient to know what region or city the user is in. It is also possible to use Gazetteers. Gazetteers are indexes of place names assosiated with an position. One can eg. search for London, and find coordinates of London. This is of course inaccurate as there is a single posiiton labeled as London, while London is in fact a quite big area. Searching for an street address however will give very accurate results if the address is found.

2.2. The Evolution Of Mobile Phones

Long gone is the time when a mobile phone was just used for phone conversations. Today's phones functions more as a small computer, organizing your schedule, reminding you of meetings, reading email and surfing the web just to mention some of the functionality common in todays phones. Functions traditionally found in separate hardware can also be found in mobile phones, such as cameras, audio recorders/players and video recorders are also becoming more common. Text messaging is widespread, and is familiar to most people, while multimedia messaging is rapidly increasing in popularity. This makes the mobile phones ideal for LBS.

GSM. GSM (Global System for Mobile communications) is the most widely used system for digital mobile phones. It supports voice, data, text messaging and cross-border roaming. Each phone in the network is identified by a SIM(Subscriber Identity Module) card in the phone. GSM is referred to as a 2G (second generation) mobile network.

SMS. The Short Message Service (SMS) is the technology that sparked the explosion in mobile phones sales. Initially a protocol for sending text via the GSM network, it can also be used for graphics (such as Nokia logos)

and ring tones. Messages is sent from one handset to another, or to a content provider returning text, logos or ring tones.

WAP/WML. The Wireless Application Protocol (WAP) is a protocol for distributing web content to wireless devices, it is the HTTP of the wireless world. WML is a markup language for such content, and is a small subset of HTML. Early WAP phones used a modem device to enable data transfer to the phone, however this was slow and expensive.

GPRS. The General Packet Radio Service (GPRS) is an additional service in a GSM network. While early WAP phones used dial up access for data transfers this service offers a package switched network service for wireless devices. This has the advantage that the handset is always online, as this is not a dial up service. Also it offers greater transfer rates, and is generally cheaper for end users. GPRS is often referred to as part of the 2.5G (2.5 Generation) wireless network.

MMS. As the popularity of SMS content grew, the need to send more sophisticated messages surfaced. The solution is the Multimedia Message Service (MMS), an infrastructure for sending and receiving messages with more than textual content. Currently MMS supports images, audio and text. As 3G networks emerge plans exist to add support for video as well. The composition of an MMS is usually, but not limited to, SMIL (Synchronized Multimedia Integration Language). Also one can add any type of content, but a receiver is not guaranteed (or even expected) to be able to handle it. SMIL is a language for making a presentation. Such a presentations consists of media in a certain order and timing. It is often the root element of a MMS message. The root element is the one describing the rest of the message. It is not requested that the root element is SMIL, but target devices are guaranteed to support SMIL. For data transfer MMS uses GPRS.

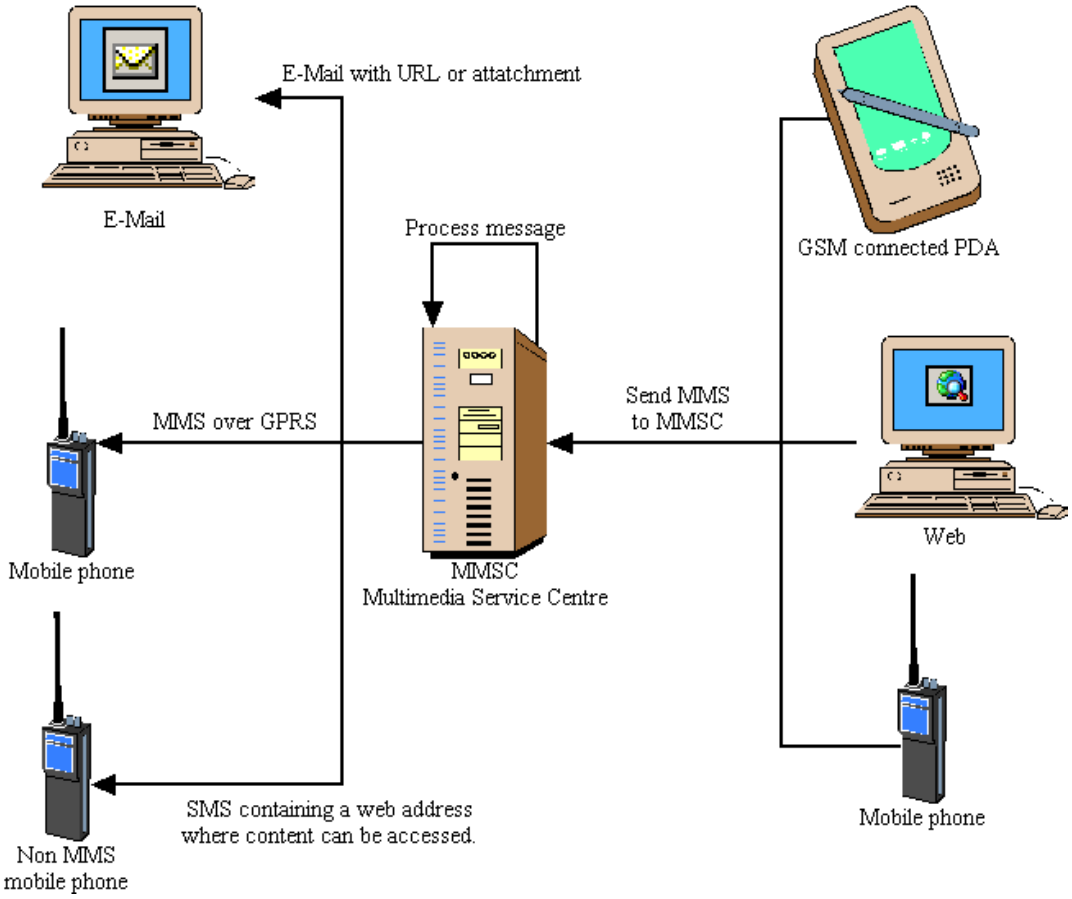

Figure 2.3. The structure of the MMS system

Bluetooth. Bluetooth is a standard for wireless communication over short distances. Originally developed by Ericsson it's now widely used by many hardware manufacturers. Bluetooth is not a proprietary technology, but an open specification. It is used in a wide range of devices such as keyboards, printers, headsets, PDAs etc. Bluetooth supports transfer speeds up to 723.2 kb/s asymmetric or 433.9 kb/s symmetric. It operates in one of three power classes, which affects the range of the Bluetooth signal. Power class 1 has a range of approximately 100m, class 2 has a range of 10m and class 3 has a range of only 10cm. These ranges may vary in different environments.

Smartphones. The term smartphones means a phone with capabilities beyond what is found in ordinary mobile phones. Smartphones run a complete operating system. Also they usually have larger displays and faster processors than ordinary mobile phones. By strict definition these devices are distinct from PDAs by having an ordinary phone keypad for input, while PDAs use a touch screen. Smartphones are however used to describe hybrid phone/PDA products such as the Sony Ericsson P900, which feature both input methods.

2.3. Symbian OS

Symbian OS is an operating system for wireless devices and smartphones, used by Nokia and Sony Ericsson amongst others. It is designed for use with phones, and has support for MMS, SMS and web browsing (WML and HTML) out of the box. Symbian devices implement an user interface on top of the OS. Nokia has its own user interface and toolkit named Series 60, while the brand independent UIQ can be found in Sony Ericsson and Motorola phones. UIQ is a pen based user interface for smartphones and also a toolkit giving developers access to telephony functions (including messaging), contacts, agenda and other phone specific features.

2.3.1. Developing for P800/P900

The P800/P900 phones run Symbian OS 7.0, and UIQ as the interface. Developers have the choice between native C++ and Java. It is also possible, with some plug-ins, to develop Visual Basic applications for Symbian[9] [43]. For Java developers there are two alternatives: PersonalJava(often referred to as pJava) and MIDP. These are both based on the Java language, but there are some key differences that make them appropriate for different applications[11] [43]. MIDP is part of the Java 2 Micro Edition (J2ME)Figure 2.4 [8], while pJava predates this specification.

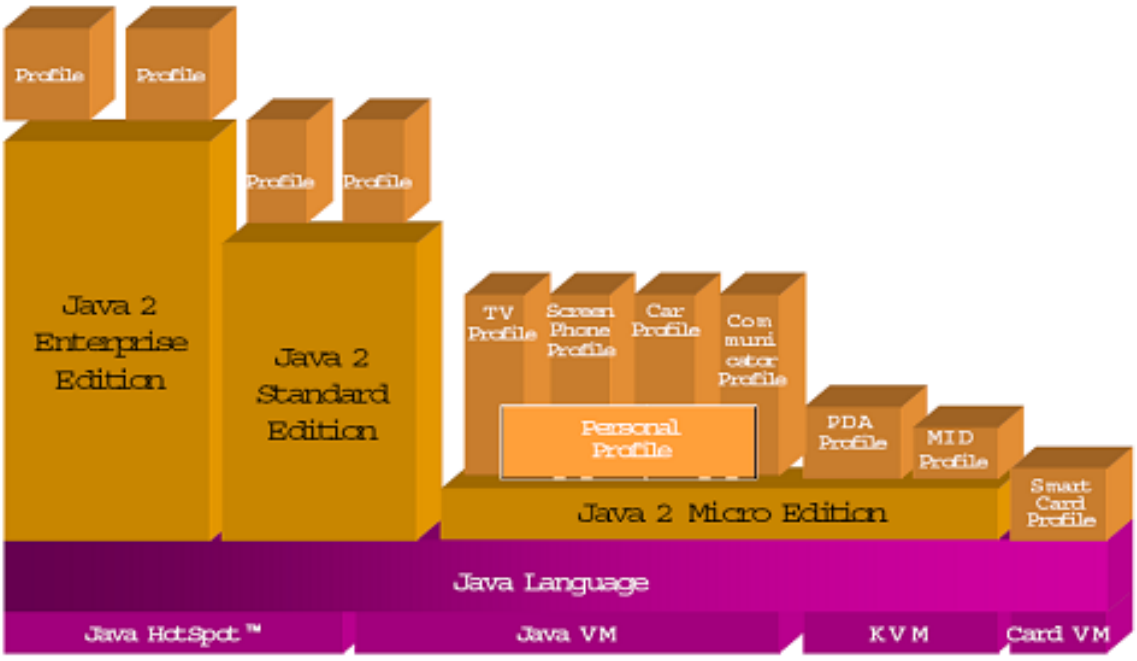

Figure 2.4. Overview of J2ME from http://www.symbian.com/events/devexpo00/pres-usa00/personaljavajavaphonepetermadany.zip

To develop for Symbian one need to install the UIQ or Series60 SDK, which can be downloaded from the Symbian website[].

PersonalJava. PersonalJava is a complete implementation of the Java 1.1 VM. While this is flexible, and lets programmers already familiar with Java get started quickly, it requires more memory and a faster CPU. Also pJava is pre-J2ME, which means it will be replaced by the Personal Profile (see Figure 2.4 [8]) in the near future. The Personal Profile is supposed to be backwards compatible with pJava though. pJava is mostly supported in expensive high end devices. pJava applications needs to be installed from a PC.

Java Native Interface. The Java Native Interface (JNI) lets developers use native libraries in their Java applications. On Symbian theese libraries are C/C++ DDLs. Loading such libraries are system dependent, but the Java syntax for doing so is not. Using native functions developers get access to functionality not usable through Java, such as messaging, call control and contacts on Symbian. Of course this means that different implementations is needed for different platforms if the application is to be portable. Also applications should behave appropreately when the library is not available.

In this project JNI is used by the GPS library. Also it will be used if MMS functionality is to be implemented. The term native implementation is used to describe the C/C++ part of a native library throughout this document.

MIDP. MIDP[12] [43] is designed for devices with limited CPU and memory recourses. It doesn't support some features one might expect from Java such as object serialization, RMI and floating point numbers. Additionally there is no access to phone specific functionality such as telephony, messaging, contacts etc. MIDP applications can be distributed 'over the air'. This makes MIDP ideal for games, as most new phones support it, and installation is easy. The P900 supports the new MIDP v 2.0 which has some additional libraries.

The Emulator. Included in the UIQ SDK is an emulator that can be used to test and debug applications. This can be faster and more flexible than having to install the application on the target device. Also you are able to see the terminal output from your program. The emulator will use your network connection and IrDA if available. Working with Bluetooth is not available though. To compile C++ (including DLLs to be used with JNI) the compiler bundled with Metrowerks CodeWarrior for Symbian OS or Borland C++ Builder is required. Compiling for the target device can be done with the GNU compiler that comes with the UIQ SDK.

UIQ Style Guidelines. In order to enforce a common look and feel for UIQ applications, UIQ has specified a set of guidelines to making UIQ applications [8] [43]. Some important principles are:

Input from stylus	The user uses a stylus instead of a mouse, and taps the screen instead of clicking the mousebutton. Common mouse operations such as double and right-clicking should not be used.
List view and detail view	UIQ applications often have a list view which present the existing data entries of the application, and a detail view giving detail on a specific data entry.
View layout	The view consists of a menubar at the top, with a menu for selecting folders on the far right. Buttons are positioned at the bottom of the view, with a 'go back button' at the far right. This button is for returning to the list view. Between the menu and buttons is the application space.
Zooming	The user should be allowed to choose the size of text.

Switch and terminate	UIQ applications should not provide an interface for quitting the application. The user can switch between applications and leave it running in the background. A spesific view should be in the same state as when the user last left. A scrollbar for example should be set to the position it was when the user last left the view. When switching to another application however the user should return to the base view (often the list view). Applications should avoid using resources when in the background, and may even be terminated by the OS if another application needs resources. It is recomended that the current state of the application is written to persistent storage when switching to another application.
Hide the filesystem	UIQ applications should not let the user see or manipulate the file system. For storage of user documents a small and flat part of the file system is visible. No system or application files should be stored here. The visible part of the file system is divided by media types; audio, video, image, document and other. The user can create folders in theese to organize documents, but no further subfolders.

2.4. XML

XML is the Extensible Markup Language. It offers a way to code data in a textual and humanly readable form. It's designed for Internet use, and is a portable way of coding data. XML itself requires instance documents to be well-formed, this makes the documents easy to parse with standard tools. It is however possible, and advisable, to further restrict the document using an XML Schema or DTD. Schemas and DTDs lets developers specify the format of documents, hence XML can be used to describe new markup languages.

An XML document begins with an definition of what kind of document it is, this is a reference to a DTD or XML Schema. When read the document is validated against its definition, and the reader is assured that the document is properly formated. The rest of the document is the data, surrounded by tags that identifies the various pieces of data in the document. Tags can be nested, and may specify attributes that affect the way the tag is interpreted.

Some of the languages specified by XML is XHTML, SVG, MathML and Docbook just to mention a few.

2.5. Scalable Vector Graphics

Scalable Vector Graphics [24] [44] (SVG) is a markup language defined in XML. It is used to represent vector graphics such as lines and shapes. It is used in the OneMap gateway to visualize map data in an web browser.

2.6. GML

GML[22] [44] is the Geography Markup Language. It is developed by the Open GIS Consortium [31] [44] OGC is a non-profit organization aiming to standardize protocols and languages for working with geografic data.

> The Geography Markup Language (GML) is an XML encoding for the transport and storage of geographic information, including both the geometry and properties of geographic features.
>
> — Description of GML [http://www.opengis.org/specs/?page=specs], OGC

GML data is sets of geographic features structured in a hierarchical manner. High in the hierachy one can find cities or areas, which is made up of several buildings and roads (geographic features lower in the hierarchy), which again is made up of geometric lines and shapes.

GML is defined as a meta-language, and it's up to developers to decide on the actual formatting of instance documents. Therefore there are several dialects of GML, with their specific sets of tools for working with them. It is common to use parts of the base GML definition for describing the geometric properties of a geographic feature, and add tags for metadata and classification of features.

Chapter 3. Design

Here I will introduce the applications that is developed. The architectural design gives detail on how the application relates to other software. Use Cases are used to illustrate how the user interacts with the application, and gives an overview of what functionality needst to be implemented. The last design phase deals with detailed design. This describes the components that make up the application, and how components interact.

3.1. OneMap Mobile Platform Architecture

The OneMap Mobile Platform is a layer between mobile applications and the OneMap Repository. The interface to the repository is WFS [23] [44], hence the Repository handles a mobile client no different from a web client. The Mobile Platform offers an API that can be used to implement different plug-ins for specific mobile devices or services. Several different applications can utilize this plug-in for interfacing with OneMap. The Mobile Platform will also be responsible for adjusting the geodata to the particular client. Mobile devices frequently have limited processing power, storage and memory, hence raw geodata is not very suitable for these devices. Also displays will vary in size and resolution, and may be monochrome.

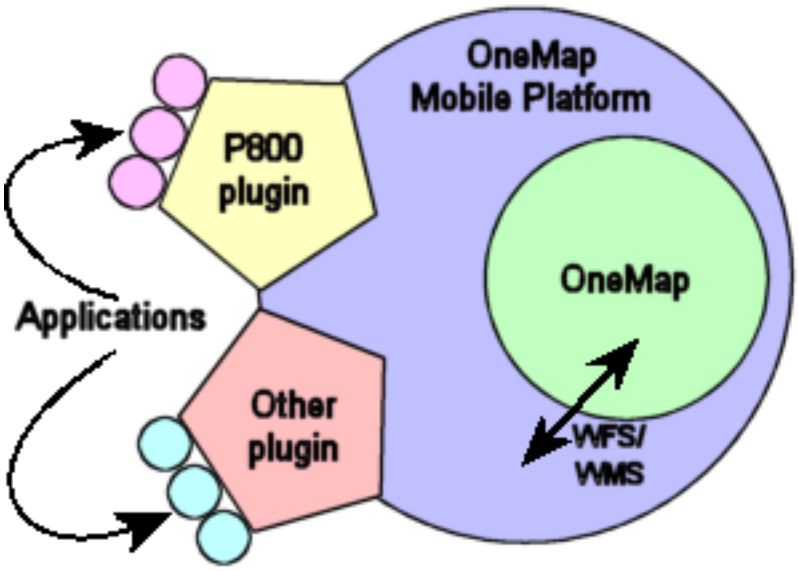

Figure 3.1. OneMap Mobile Platform Architecture

The P800 Plug-in. This plug-in uses MMS as the content protocol. Although our target platform is the P800 and P900 phones, the use of MMS enables several phones to use at least a subset of this plug-in's functionality. A GSM/GPRS modem will be used to receive MMS-es. The MMS will be decoded, and the media content will be stored as a geographic feature in the Repository.

3.2. Use Cases

Before looking at the program design, it is important to consider what functions the application is to provide. UML Use Cases[] is used to see the application from the user's point of view. It describes the functions the user

is offered, and is a good starting point for designing the user interface. Also it's easy to identify the functionality that needs to be implemented. The use cases for BTDT, which is the central application in our system will be described first. Other applications implemented will be described later.

3.2.1. Been There - Done That

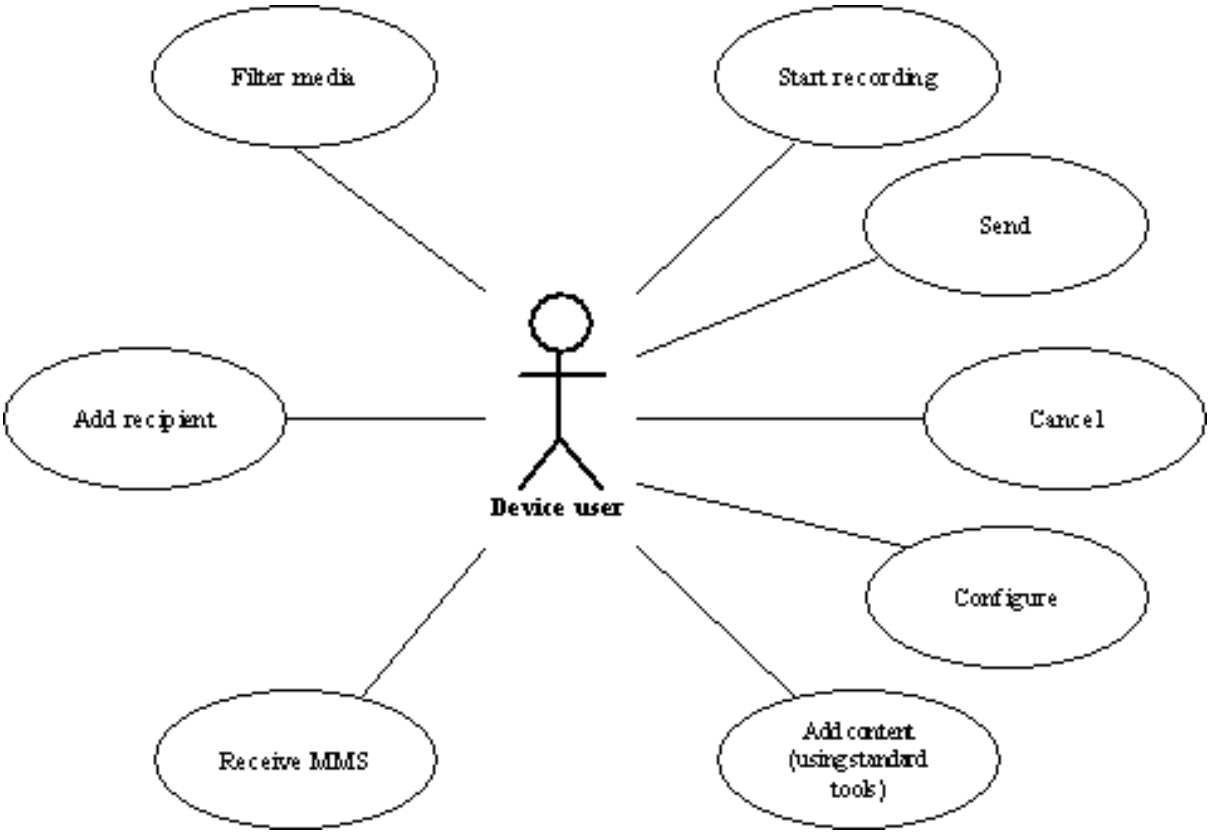

Figure 3.2. Main use case for Been There - Done That

This illustration shows how users may interact with the BTDT application. The use cases to the right of the actor is essential to the application, while the ones on the left are possible additions which will only be implemented if time allows.

Start recording. This lets the user start the recording of a message. This will save the current time, and change the status of the application. The GUI will indicate that the application is recording. This use case can only be initiated when in ready state.

Send. This use case is initiated when the user has finished adding the content he wants to send. Once again the current time will be recorded, and the media created in the period between 'start recording' and 'send' will be identified. This media will be added to a new MMS message, and then sent to a specified recipient. The GUI will indicate that a message is being created and sent, before indicating that the application is in ready state again. This use case can only be initiated when in recording state.

Cancel. This use case is initiated if the user decides he doesn't want to send the message he started creating. It can only be initiated when in recording state, and will change the application to ready state.

Configure. This use case lets the user change how the application behaves. A new GUI will show up. Here the user is given the opportunity to select the locations and types of files to search for when the Send use case is

initiated. Also the user can set the network protocol options.

Add content. This use case doesn't actually relate directly to the BTDT application in a programming context, but it does relate to the usage of the application. The user will add media using the standard tools on the phone, which already should be familiar for the user. BTDT should be in the recording state for this use case to be initiated. BTDT will not be directly affected when this use case is initiated, as it is only running in the background. See Figure 3.4 [16] for a more detailed description of the various use cases covered by this use case.

Filter media. This use case lets the user specify what media to include in a message when the send use case is initiated.

Add recipient. Lets the user add a recipient that will receive the message. Should preferably allow more than one recipient.

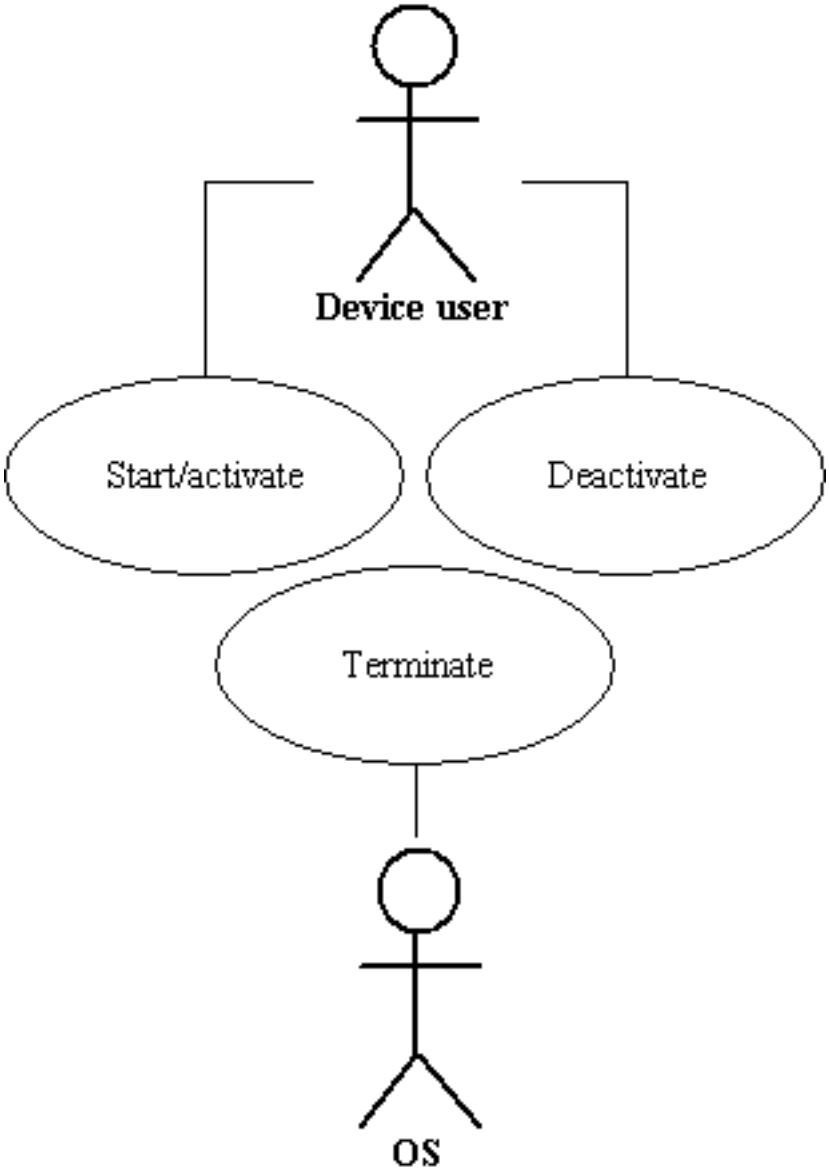

Figure 3.3. Use case for load, switch and terminate

This figure shows functions related to program loading, switching and terminating. This kind of functionality is

an important part of the UIQ Style Guidelines [8] [43].

Start/activate. This use case is initiated when the user taps the BTDT icon. BTDT may not be running, or it may be running in the background already. It may be in any state. A status file is read, and the GUI is updated accordingly.

Deactivate. This use case is initiated when the user switches to another application. The current status of the application is written to file.

Terminate. This use case is initiated when the phone is shut off. Also the application can be closed by the OS if it needs resources.

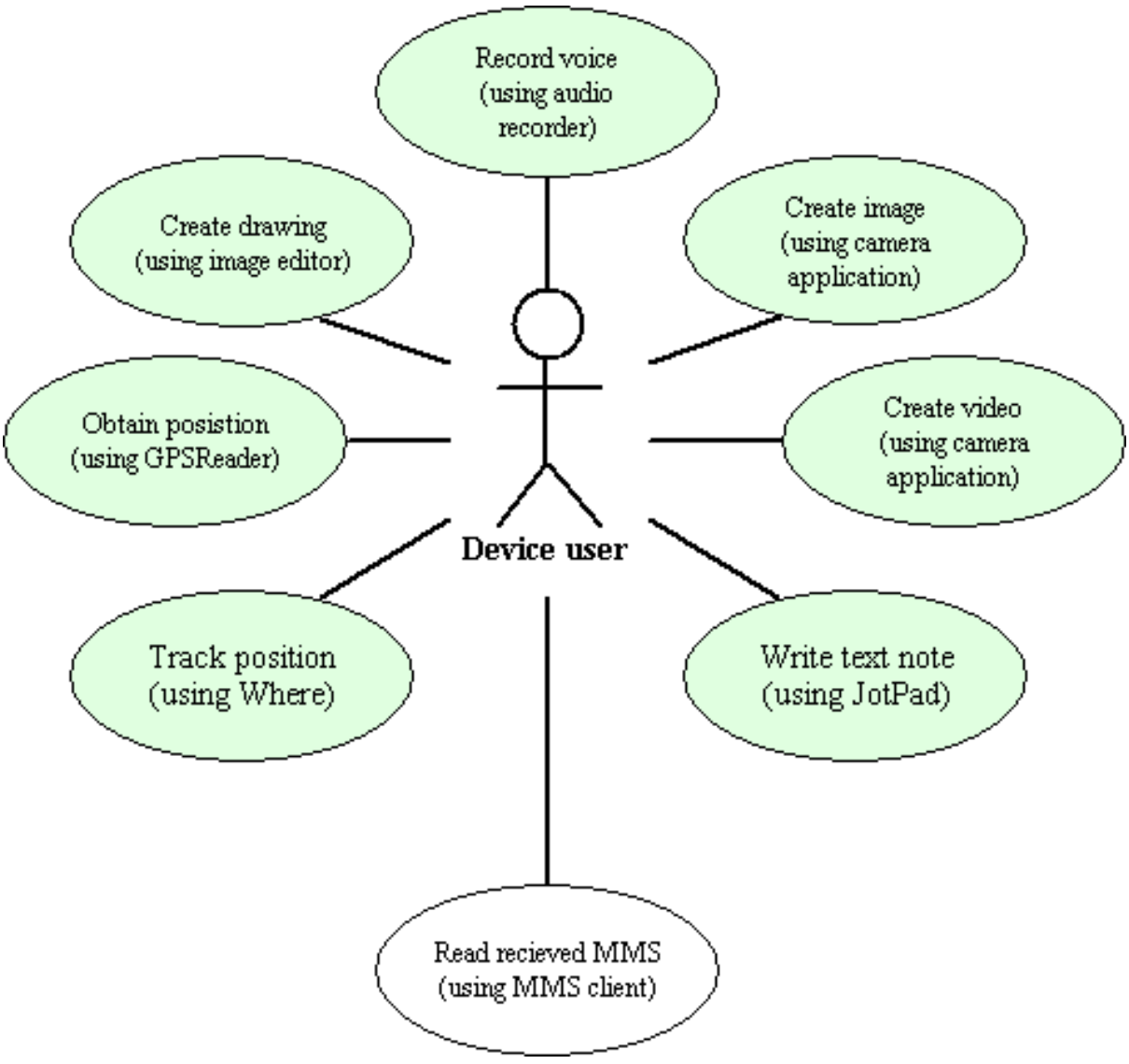

Figure 3.4. Use cases for the tools surrounding BTDT

Even though creating media isn't handled by the BTDT application, it is an important use of the application. Some applications for creating media is even implemented for this specific purpose, such as Where. The use cases colored green are all related to the 'Add content' use case described earlier. The following use case descriptions include an description of the application one would use on the P900 for that specific type of media.

The 'Read received MMS' use case is not directly relevant to BTDT, but may be a subsequent result of using it.

Record voice. To add voice media to the message the user uses the 'Sound Recorder' tool. This application is not consistent with the UIQ style[8] [43] principle of list/detailed view. Also it hides its recorded files from the user instead of saving them in the 'Media Files' folders. The files are saved in 'documents\Voice\VoiceNote' as a numbered .wav file. For the user it can be used as normal though, as BTDT will be able to identify new audio files.

Create image/video. The CommuniCorder application (named CommuniCam on P800) lets the user take pictures and record video. Pictures are saved in JPEG format in any folder of the users choice under 'documents\Media files\Image'. Video is saved in .mp4 format under 'documents\Media files\Video'.

Write text note. Symbian includes an application for writing notes and drawing sketches, called 'Jotter'. Unfortunately this application is not fitting for our purpose. It does not store the notes as text files in any user-visible part of the file system. Rather it stores all text and sketch notes in a single file. This means that BTDT would have to be able to read Jotters file format. Also it is not consistent with with our scheme of adding new files to the message. To compensate for this a simple application, 'JotPad' will be implemented.

Create drawing. As mentioned above, Jotter is not appropriate for our application. Another, however more cumbersome, approach for drawing pictures exist, and will serve its purpose for BTDT. To make a drawing the user needs to do the following:

- Open images
- Open an image in the list
- Tap Edit -> Edit picture
- This will open an editor quite similar to Jotter. A completely white picture should be stored, to be used as a blank drawing. When exiting returning to the list view, the user is asked to save the drawing. This file will be recognized by BTDT.

Obtain position. For communicating with the GPS device a separate application is implemented. This is consistent with the scheme of using external tools for creating media. Also it makes it easy to implement positioning using other methods than GPS. A position is a set of coordinates (latitude and longitude) and the time the position was acquired.

Track position. A track is a set of positions, ordered chronologically. To obtain a track one would read the GPS at a given interval until. This enables the user to e.g. walk a path, and save it as a geographic feature. Both use cases relating to the GPS is handled by Where.

Read received MMS. This use case is initiated when receiving an MMS as a response to a request sent to the OneMap repository. BTDT will not implement any functionality for handling this use case.

3.2.2. Where (the GPS application)

Adding location data to a message is handled in the same way as any other media. A separate application for connecting to the GPS is therefore implemented. Tracks and positions will be saved as files in GPSml[18] [43] format.

Where will follow the UIQ Style[8] [43] principle, by implementing an list view, and a detail view.

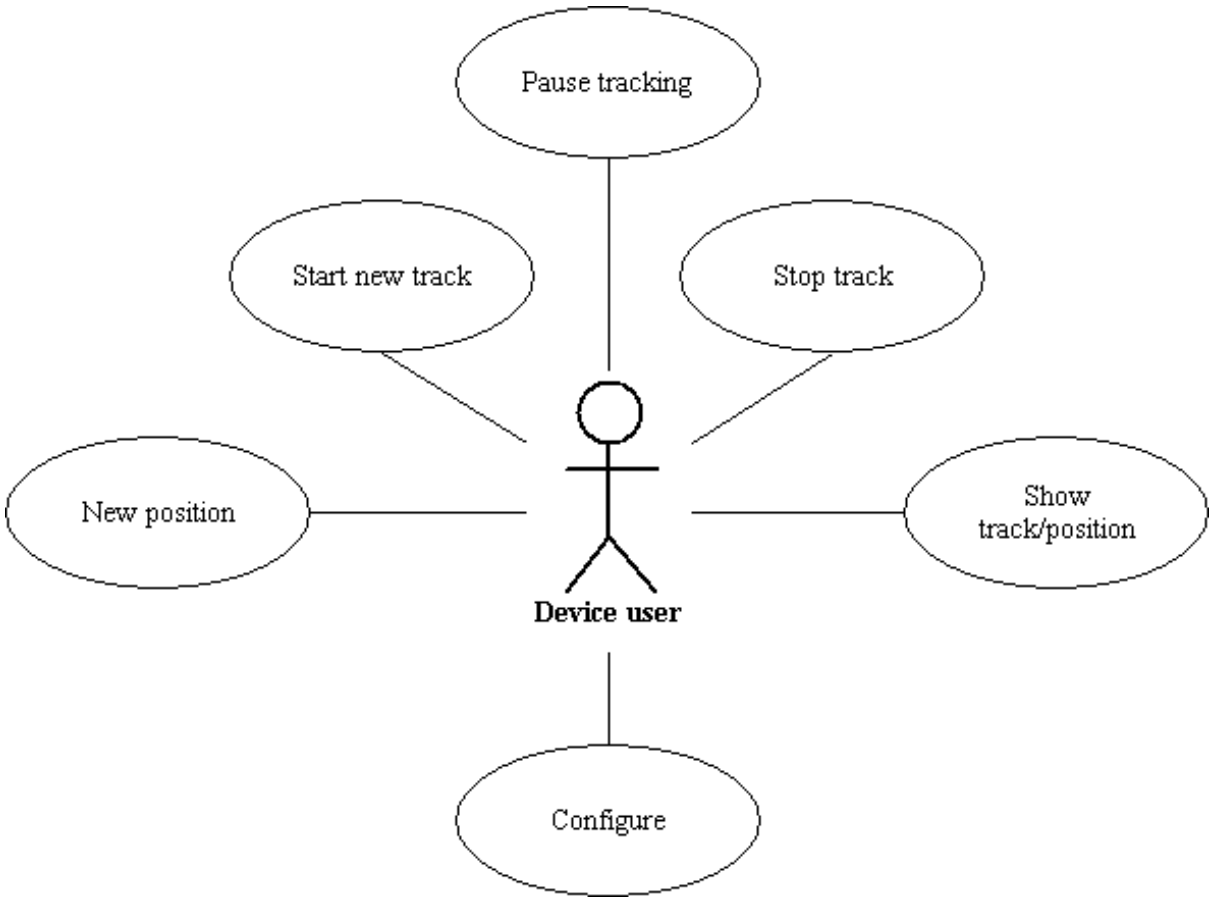

Figure 3.5. Use cases for Where

New position. This use case is initiated from the list view. When initiated the position is read from the GPS device, and stored as a where file. The user will be prompted to enter information about the position.

Start new track. This use case is initiated from the list view. It prompts the user for information on how to record the track, or to accept the defaults. The application will continuously read from the GPS device at certain intervals, and add the position to the track.

Pause track. This use case can only be initiated when tracking. The user may want to pause the track e.g. if he will remain at the same location for some time. This use case can possibly also be initiated by the application if it should lose contact with the GPS device or the device looses contact with satellites, to prevent tracking of lacking or false data.

Stop track. When initiated the track will be saved to a file, and the user will be returned to the list view.

Show track/position. This use case lets the user view an already created position or track.

Configure. This will open a configuration view, which lets the user enter default values for track settings, such as how often to read the GPS when recording a track. [MORE DETAIL HERE]

3.3. Detailed Design - Been There - Done That

Been There - Done That is implemented in PersonalJava[28] [44] . It is designed to be able to use different communication protocols.

The class diagram Figure 3.6 [19] shows what classes the application is composed of. The different components will be described in more detail below.

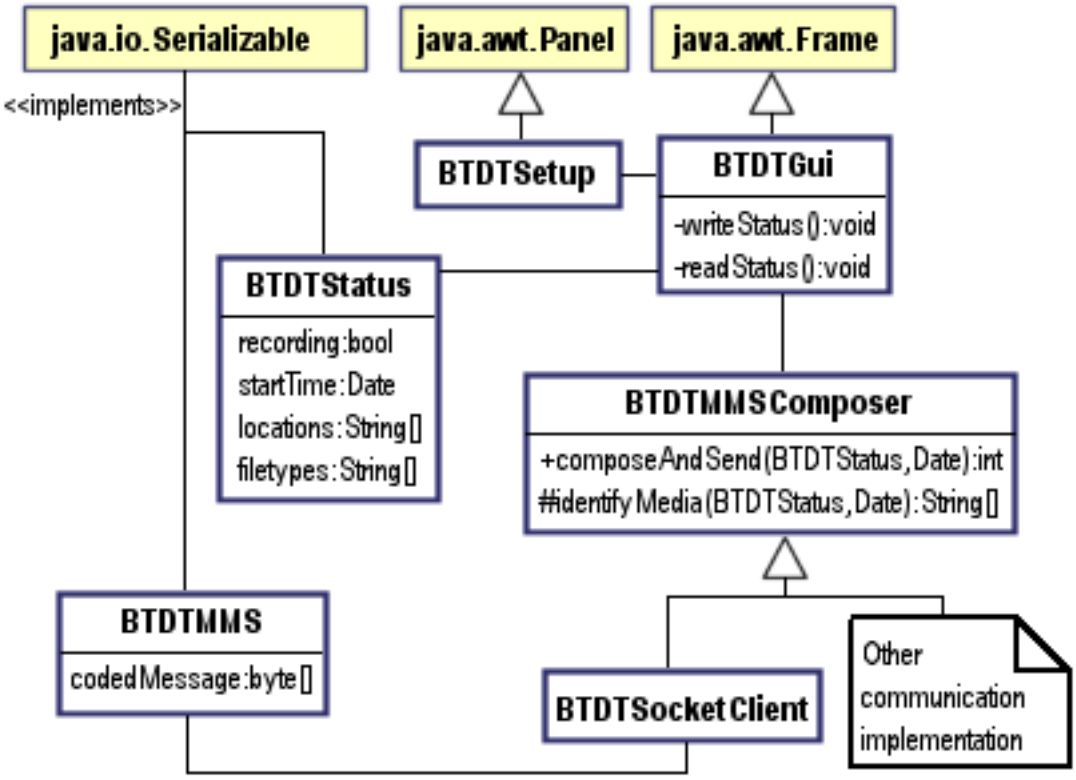

Figure 3.6. Class Diagram of BTDT

BTDTGui. This class is the central class of the application. It defines the main GUI, and binds the other classes together. It is responsible for reading status on startup and storing status when the application is deactivated or terminated. It also handles input in the main GUI.

BTDTStatus. This class holds all information on the current status and setup of the application. It implements the java.io.Serializable interface and can be written to disk when the application terminates. When recording is started the class member recording is set, and the current time is stored in startTime. BTDTStatus also keep a list of locations (in the file system) to search for media, and a list of file types to include when searching.

BTDTMMSComposer. When the user initiates the Send use case, all the actual work is done here. The method *identifyMedia* searches the locations specified in the status object for media, and is used by composeAndSend to identify new media files. The method composeAndSend is declared abstract, and i implemented in subclasses. This lets the application use different communication protocols a uniform way.

BTDTSocketClient. This is an implementation of the client side of the communication protocol described in Section 4.4.4, “Alternate Networking Implementation” [30]. The coded MMS is wrapped in an BTDTMMS object, and transmitted on an socket connection to the server. The method identifyMedia implemented in the superclass is used to find the media files to add to the MMS.

BTDTMMS. This is a wrapper for the encoded MMS generated using the Nokia MMS Library[29] [44]

BTDTSetup. This class is a GUI for configuring how the application will function. If the user chooses to save

the new configuration the status object is updated. This class handles its own input.

3.4. Detailed Design - Where

Where is the application used for communicating with the GPS device. It saves positions and tracks in GPSmll format (see Section 4.3, “XML Schema for GPSml lite” [28]). Below more detail on the different classes is given.

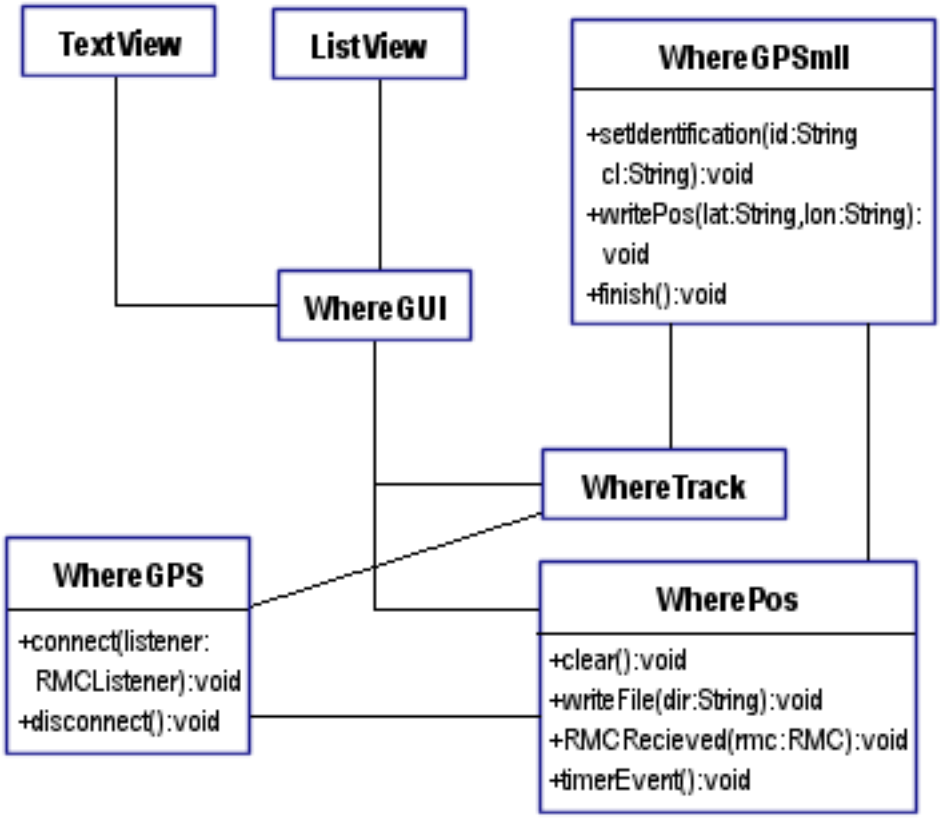

Figure 3.7. Class diagram of Where

WhereGUI. This is the central class of this application. It defines the main GUI, and binds the other classes together. It handles input from the GUI.

ListView (from package arneehan.widgets). This is a GUI widget implemented to resemble the UIQ list view. It identifies files present in the selected directory, and sends a FileSelectionEvent to the specified listener. In this application WhereGUI listens, and opens the selected file in a TextView widget.

TextView (from package arneehan.widgets). This is a simple GUI component for working with text files. It supports reading and writing text files, and is used in this application to display the contents of a GPSmll file.

WherePos & WhereTrack. These two components define GUIs for recording a position and a track respectively. The component communicates with the GPS device using the WhereGPS class, and writes the result as a GPSmll file using the WhereGPSmll class. Both handle their own events.

When trying to connect to the GPS device, the GPS might not be turned on, or something else prevents the

devices to establish a connection. To prevent the application from behaving incorrectly when this happens a timeout function terminates the attempt after a certain amount of time. The application uses a class Timer (from the package arneehan.util) for this purpose.

WhereGPS. This class functions as a utility layer between the WherePos and WhereTrack classes, and the GPSLib communications library Section 4.2.1, “GPS Library” [27]. It specifies two methods, connect and disconnect. Connect takes an RMCListener that events will be dispatched to when receiving data from the GPS. Disconnect closes the Bluetooth connection.

WhereGPSmll. This is a utility class for writing position data in GPSmll format. Though not an XML implementation it does enforce some guidelines that will secure that the file written is proper GPSmll by throwing IllegalStateExceptions when something illegal is attempted.

3.5. Communications Protocol Design

To transmit data from the device to the OneMap server MMS is used. MMS is integrated in the GSM network, and uses GPRS for high speed data transfer. Also using MMS means that some functionality will be available on phones with no specific hardware at all. On the server a GSM modem accepts MMS messages, decodes the contents and stores it for future retrieval. The messages will not be formatted in any particular way. It may prove necessary to split large messages into several pieces. Location content is handled in the same way as other data. Each position or track is sent in a .gps file. The format of the gps file is specified in an XML Schema described in more detail in Section 4.3, “XML Schema for GPSml lite” [28].

Chapter 4. Implementation (I should customize this title - or maybe I shouldn't)

First in this chapter I will introduce the programming environement used when making Java applications for Symbian. This complements the more theoretical introduction to programming for Symbian in the chapter []. In this chapter the practical issues are discussed.

Next I introduce third party libraries used in the implementation, specifying, to a certain degree, how they work internally, as well as how they are used in this application.

About the actual implementation I have given details on the issues that I found interresting. This part is complemented by the corresponding sections under design and improvements. For even more detail see the [javadoc] and the actual sources[].

4.1. Programming Java For Symbian In Practice

To develop for Symbian one first needs to obtain a SDK for the target platform. SDKs available are; Series60 for Nokia phones, UIQ v2.0 for the P800 and UIQ v2.1 for the P900. For programming Java there doesn't seem to be any difference between versions 2.0 and 2.1. At least I've been able to run applications developed on 2.0 on the P900 and applications developed on 2.1 on the P800.

The next thing one should install is the Java 1.1.8 Development Kit (JDK). This is the most recent of the Java 1.1 versions. Java applications is compiled with the javac compiler included.

4.1.1. Using the Emulator

It may be desirable to run the java application in the emulator when building the GUI, debugging etc. In order that the emulator can access the compiled program parts of the Windows file system needs to be added to the emulators file system. This is done setting shell variables. Eg. the command

```
set _epoc_drive_j=%SDKPATH%\epoc32\Java
```

will map the path after the = to drive J on the emulator. This location should be added so that the emulator can find the Java API classes. %SDKPATH% refers to the path where the SDK is installed, eg. C:\Symbian\UIQ_21

The program can be run using the following command:

```
%SDKPATH%\epoc32\release\winscw\udeb\pjava_g -cp K:\Hello Hello
```

The -cp option adds K:\Hello to the emulators classpath. Of cource drive K needs to be defined as above. Hello is the name of the class to run, and pjava_g is the executable that launches a Java application in the emulator. If drive K is mapped to C:\Symbian\src, the above command would correspond to:

```
java -cp C:\Symbian\src\Hello Hello
```

4.1.2. Deploying Applications

When the application is tested in the emulator one might want to run the application on the target device. The SDK comes with tools for preparing the application. This description is for Java applications, the tools and methods are quite simmilar for other applications.

First the application must be compiled with the javac compiler. The class files can be added to a java archive (JAR) file, with the command:

```
jar -cf dist\Hello.jar *.class
```

Applications can not be started from a command line on Symbian. Applications are started from the Launcher application. Launcher needs some information on how to start the application, and an icon to display in the Launcher. This information can be generated with the aifbuilder tool, which is started from the command line with the command:

```
aifbuilder
```

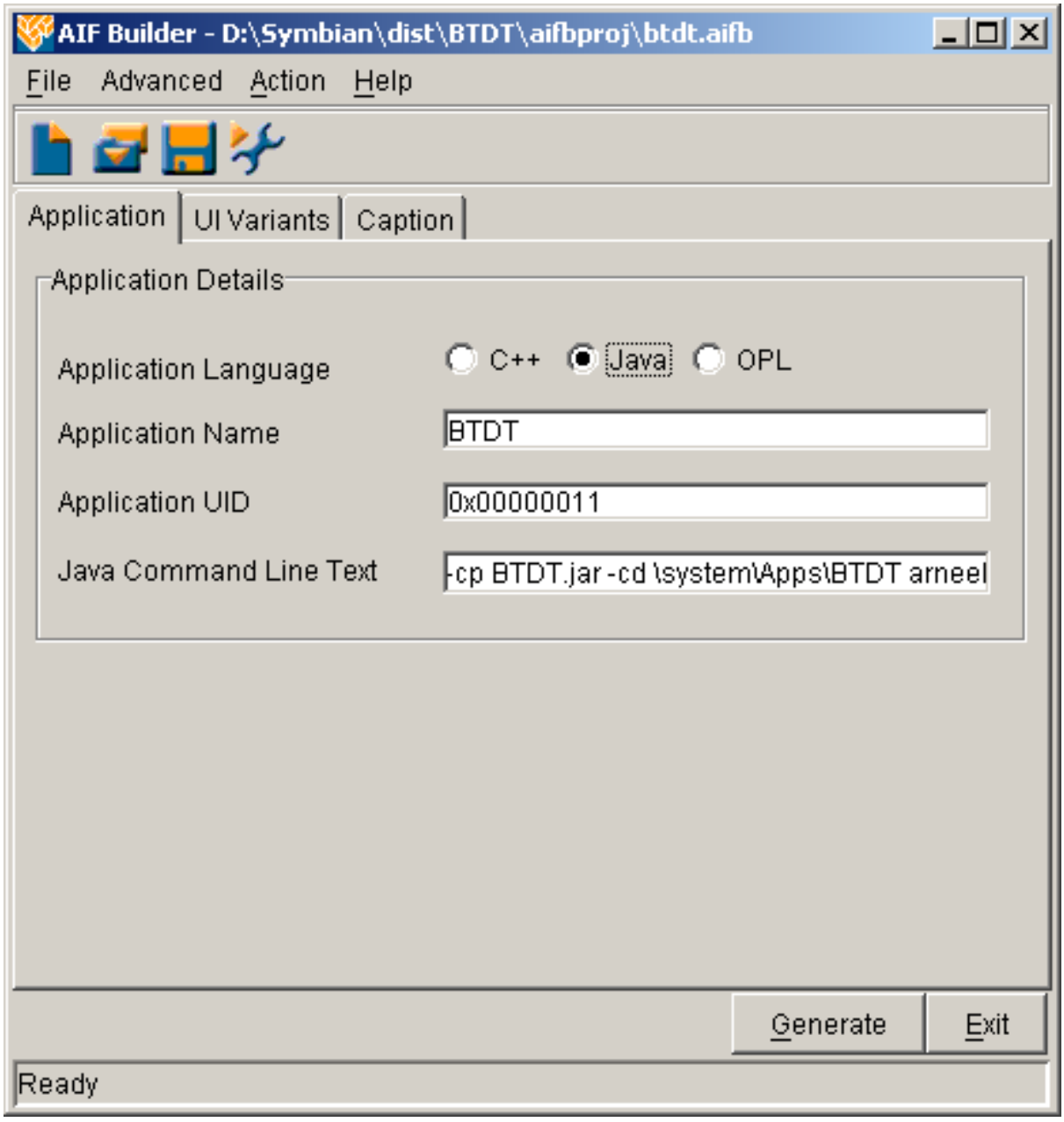

Figure 4.1. The aifbuilder application, Application tab

Aifbuilder is a GUI program. After starting a new project (see Figure 4.1 [25], one should select Java, enter a name for the application, an UID and the command used to start the application. UID is a unique global identifier assigned to every Symbian application. UIDs must be obtained from Symbian to ensure that it's unique. For development however one can use values between 0x00000001 and 0x0FFFFFFF. These values are reserved for developing, but should never be used if the application is distributed. The command specifies the classpath (-cp), the current directory (-cd) and the class to run. The current directory needs to be set as the Launcher application don't have an assosiated current directory.

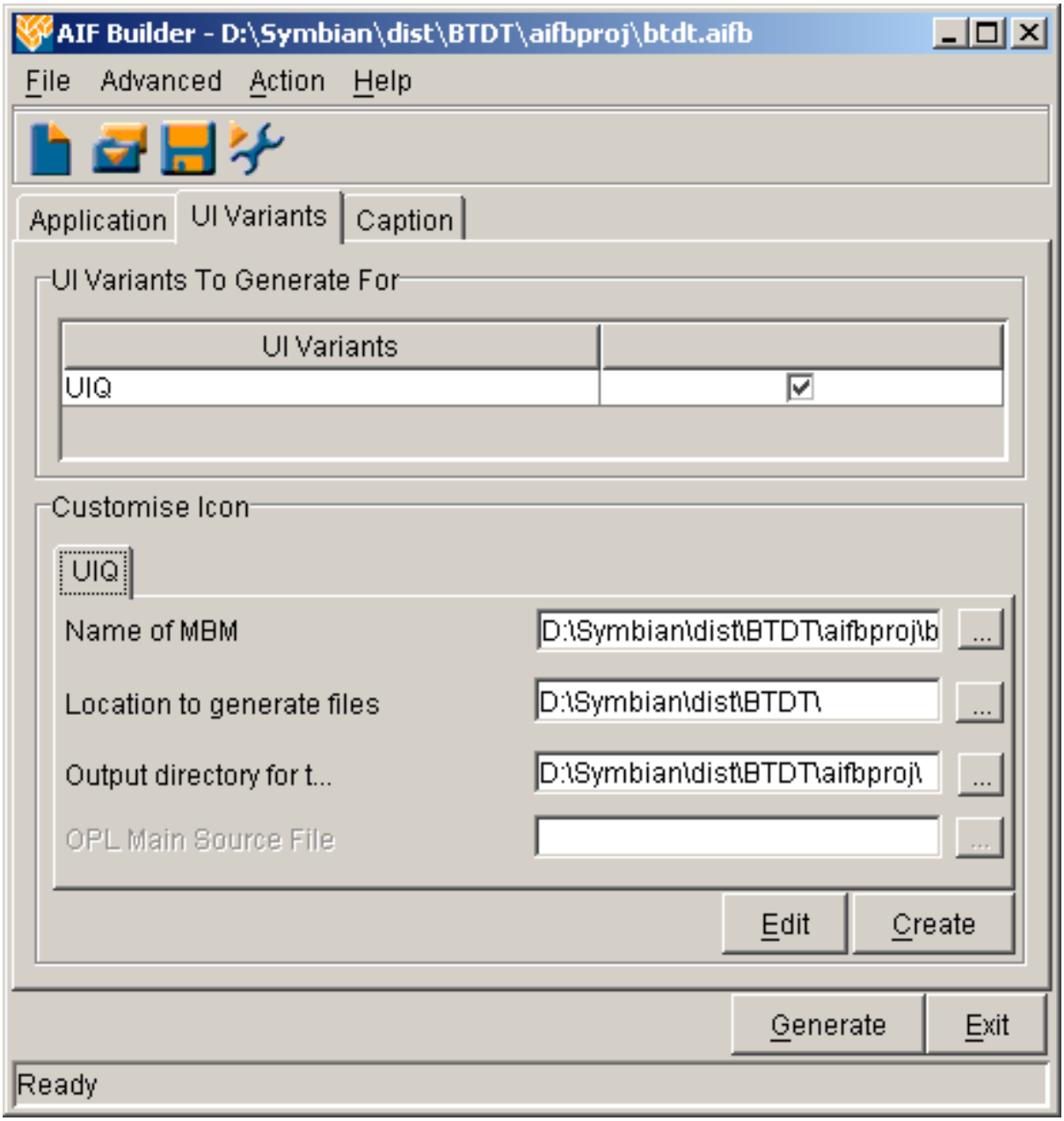

Figure 4.2. The aifbuilder application, UI variants tab

Switching to the UI variants tab (see Figure 4.2 [26]), make sure UIQ is selected. The 'name of MBM' field specifies an icon or one can be made with the 'create' button. The other fields specify where output is to be saved.

To install programs Symbian uses the Symbian Instalation System (SIS). To distribute programs the only thing needed is a .sis file. To make such a file one needs to specify what files to add to the package in a .pgk file.

```
#{"Where"},(0x00000111),0,3,0
(0x101F80BE), 1, 0, 0, {"SonyEricssonP80xPlatformProductID"}
"Where.jar"-"!:\system\apps\Where\Where.jar"
"Where.app"-"!:\system\apps\Where\Where.app"
"Where.aif"-"!:\system\apps\Where\Where.aif"
```

```
"Where.txt"-"!:\system\apps\Where\Where.txt"
"infoprint1.dll"-"!:\system\libs\infoprint1.dll"
```

Figure 4.3. An example package file for the Symbian Installation System.

The first line identifies the package with the name "Where" and UID "0x00000111". These values should correspond with the ones entered in aifbuilder. The last three numbers are version information (major, minor, build). The second line specifies the P800 as the target platform for the application. This works fine with the P900 as well. The rest of the lines specifies which files the package is composed of, and their location on the target device.

Finally the .sis file can be created with the command:

```
makesis Hello.pkg
```

This generates Hello.sis which can be used to install the application on the target device.

4.2. Libraries Used In The Implementation

The implementation uses some Java libraries in addition to the standard JFC.

4.2.1. GPS Library

This library takes care of communication with the Bluetooth GPS device. It is implemented in C++, and made available to Java using the Java Native Interface. More information from the author can be found in [16] [43], and an minimal example of using the library can be found in [17] [43] The Java part of the library is composed of the following classes:

GPS. This is the interface for programmers using the library. It specifies the methods startReading, stopReading and addRMCListner. When the GPS returns data RMCRecieved events is sent to the RMCListener.

GPSReader. A layer between the GPS class and the native implementation.

InfoPrint. This specifies the interface of the native implementation. The actual implementation of these classes are provided in InfoPrint1.dll.

LatLon. Utility class for representing the a latitude/longitude pair.

RMC. Utility class for representing the data received from the GPS device. Fields include position, speed and fixTime.

RMCListener. This interface specifies any implementing class as a RMCListener. The method RMCRecieved is called, and passed a RMC object, when a string has been received from the GPS device and parsed.

4.2.2. Nokia MMS Library

This is a library supplied by Nokia for working with MMS[29] [44]. It is originally designed for Java 2, but with only minor modifications I was able to compile for pJava. It contains classes for creating MMS, adding files to an MMS, coding MMS, sending MMS etc.

4.3. XML Schema for GPSml lite

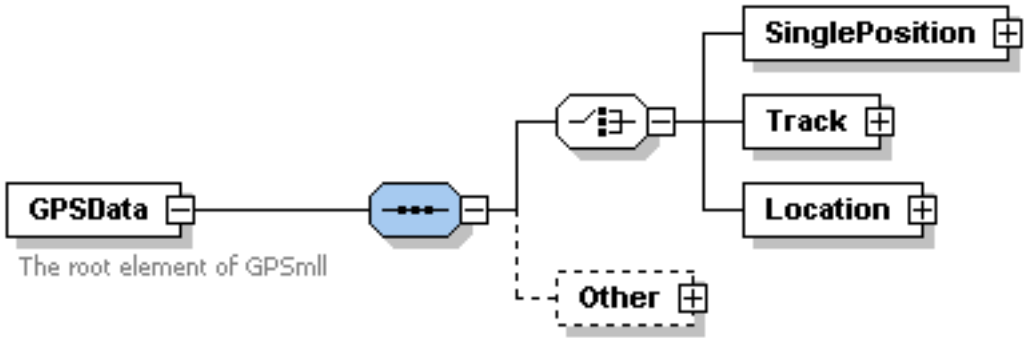

Figure 4.4. GPSml lite: Overview of GPSml lite

GPS positions and tracks are stored in single files as XML. The format of the XML file is specified by an XML Schema. The schema is a small subset of GPSml [18] [43], tailored for our particular use. We've tried to make the schema as simple as possible while maintaining conformity with the original GPSml. There is no Java API for working with XML on the phone, and even if we introduced one the limited recourses available endorse a simplistic design.

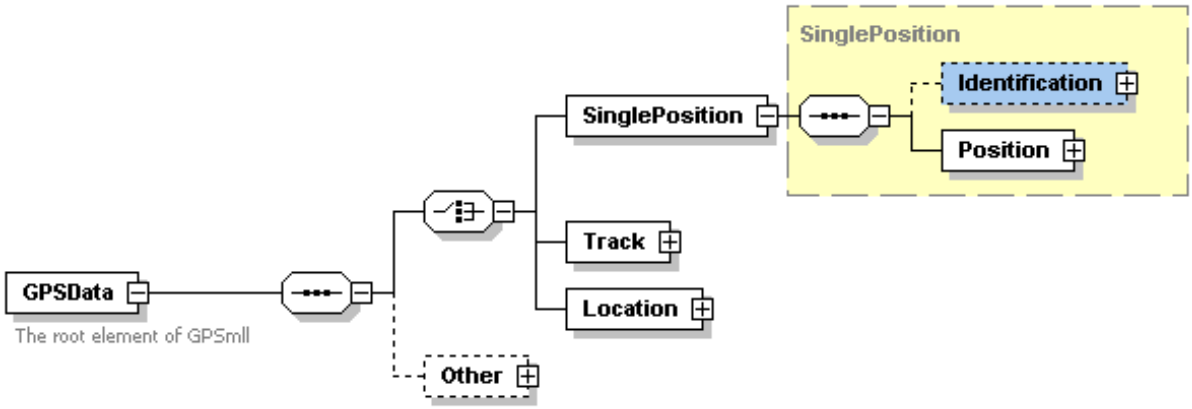

Figure 4.5. GPSml lite: Detail of the SinglePosition tag

As in GPSml the root element in GPSmll is GPSData. GPSData must contain exactly one of the types SinglePosition, Track and Location. SinglePosition represents one position, and can optionally be identified by an Identification. A track is a series of two or more positions. Also here additional information on the track can be specified in an Identification tag. Location represents a textual position, such as an address or place name. The Location tag makes the format possible to use even if a GPS device is not available.

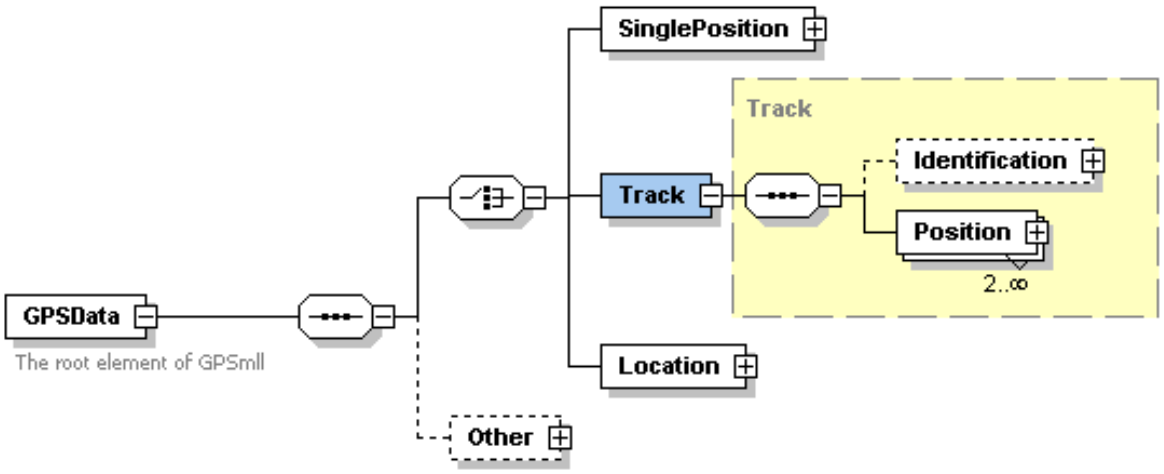

Figure 4.6. GPSml lite: Detail of the Track tag

Position, Identification, Location and Other are unmodified parts of the GPSml schema, while the rest is implemented for this particular use. The original GPSml allowed several instances of data in the same file under the Collection tag. In GPSmll only one is allowed.

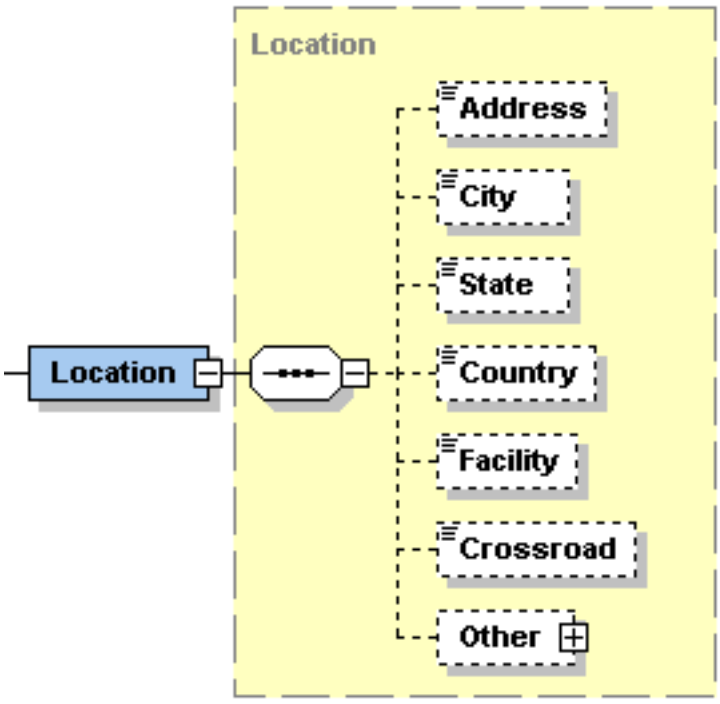

Figure 4.7. GPSml lite: Detail of the Location tag

4.4. Implementation of BTDT

Been There - Done That is implemented in pJava. See Section 3.3, “Detailed Design - Been There - Done That”

[18] for an overview of the classes that constitutes the application.

4.4.1. Debugging

When an application is running on the phone there is no standard output to write debug messages on. To make it possible to debug on the phone I added an TextArea GUI component to the application that is used to write debug messages. This should be removed in a final product, but comes in very handy at this stage in development.

I also use a class Debug to filter debug messages. Using different subclasses of Debug, messages is sent to different targets (standard out, text component, file).

4.4.2. GUI Considerations

When working on a small screen like that on a mobile phone, it is important to use the space effectively. The Button class in the AWT only support text labels. This however is undesirable on small screens, as buttons can take up a lot of space. A new button was therefore implemented that allowed graphics or text on the label. Currently it's limited to a set of predefined icons though. These icons are stop, pause and record.

The main GUI of the application presents the functionality that is needed to create a single message; record button, send button and text fields for setting receiver and subject of the message.

The Setup GUI lets the user select what locations to search for media and what types of media to search for. This approach is useful for development, but should not be used in a final product. It's an important Symbian principle that the user is not supposed to see the underlying file system, and also it is not very user friendly. It should probably be substituted with checkboxes letting the user specify which types of media to be added.

As far as possible layout managers is used to let the application adapt to different screen sizes, and the application should be usable on other target platforms. If the screen is very small some components may not be visible or usable.

4.4.3. Identifying New Media

New media is identified by searching a set of predefined locations for certain file types, and checking the 'last modified' attribute of the file. If this is between the start and stop times the file is included in the MMS. A problem with this approach though is that the lastModified in java.io.File is inconsistent. The documentation says the following:

> Returns the time that the file represented by this File object was last modified. The return value is system dependent and should only be used to compare with other values returned by last modified. It should not be interpreted as an absolute time.
>
> —java.io.File - Java 1.1.8 Documentation[6] [43]

This means that there is no guarantee that the value returned can be compared with the time returned by getCurrentTimeMillis() in java.lang.System, which is what we intend to do. On my computer running Windows 2000 this is not a problem, but on the P900 the time returned by lastModified does not take the current time zone into consideration. Hence it will only work if the time zone on the phone is set to GMT. This will also be a problem if the application is to be ported to another platform, even other Symbian devices may not handle this in the same way.

4.4.4. Alternate Networking Implementation

To avoid the complexity of implementing an MMS client using the GSM modem, we implement an alternative solution for communication between the device and the server. We will utilize the Nokia MMS library to

compose and encode the MMS. We use a Java socket based connection for transmitting the encoded MMS. Sending data back to the phone is more difficult with this approach though. The handset is given an IP when opening a GPRS connection, which is unknown to the server and may be different each time the handset makes a connection. For more info on GPRS connections see [19] [43].

4.5. Implementation of Where

Where is also implemented in PersonalJava, and is using GPSLib for interfacing with the GPS device.

4.5.1. GUI Considerations

The user interface is made to resemble the UIQ Style[] as far as possible. Java widgets have a different look than the native widgets, which makes it difficult to stick to the style guidelines without implementing several new widgets. When implementing Where a compromise between the resulting style and the time spent making the GUI must be made.

A ListView widget is implemented to resemble the standard UIQ list view. It is implemented as a reusable component, and can be fitted to other uses. The actual list is an instance of java.awt.List. In addition to handling file selection from the list, it can build a menu to select a location in the file system. This is the menu seen at the far right on the menu bar in UIQ applications. This hides the file system from the user. The only visible parts is a flat set of folders in e.g. Media files/image/. Also the user has a choice between internal(phone memory) and external(memory stick) storage.

4.5.2. XML in Where

Currently there is no real checking of the generated XML document for validity or well-formedness. The WhereGPSmll class enforce some control on the output, and GPSmll is a simple language. However no checking against the schema is done, and the generated document may contain illegal values. Parsing documents on the relatively slow processor and limited memory on the P900 is undesirable. If however this is implemented a SAX based parser should be used, as this uses less memory than the DOM based parser. There is no support for XML in PersonalJava without adding third-party libraries. For just checking well-formedness one could use XML Light[30] [44]. It is a simple parser, which runs on Java 1.1.

4.5.3. Working With The GPS Library

As of the current version all error checking in the GPS library is absorbed in the library itself. This gives the programmer little opportunity to deal with errors in a graceful way. To compensate for this one would have to change some aspects of the library, or code closer to the native implementation. Another possibility which I have tried is to cancel the connection if no data has been received in a certain time. This have not been a successful approach though as the procedure for cancelling the connection doesn't work properly.

I have made som changes to the native implementation to fix a bug that prevented the connect method from returning an error code when a connection could not be made. As a result the application can detect this error, and doesn't cause what I believe is a segmentation violation from trying to read from a connection not in existence.

The function for closing the connection does not work as espected. After calling this method I'm not able to make a new connection without restarting the application. I have not been able to rectify this problem.

Chapter 5. Results

Here I will discuss the solution reached. First I will schetch out what functionality is implemented, and present the applications user interface. In the second part I will compare the solution with what was intended when the project was started, and present some improvements that can and/or should be made. Last detail will be given on how to install and run the application on a P800 or P900 phone.

5.1. What Functionality Is Implemented

5.1.1. Been There - Done That

BTDT is implemented as specified, using the alternate communications protocol defined in Section 4.4.4, “Alternate Networking Implementation” [30]. Media files is identified and sent using a socket connection to the server. The user may specify a receiver, but this is ignored. The MMS is always sent to the server. The user also have to specify the from field. This is usually the phone number of the sender, however no functionality has been implemented for checking the handsets phone number.

Figure 5.1. Screenshot of the main GUI of BTDT

This screenshot shows the main user interface of BTDT. The big text area is only used for debugging, and should be removed in a final version. The buttons are 'setup', 'start recording', 'stop recording' and 'send recorded message'. The 'setup' button brings up the screen seen in Figure 5.2 [34]. Here the user can select what file types to include in the message.

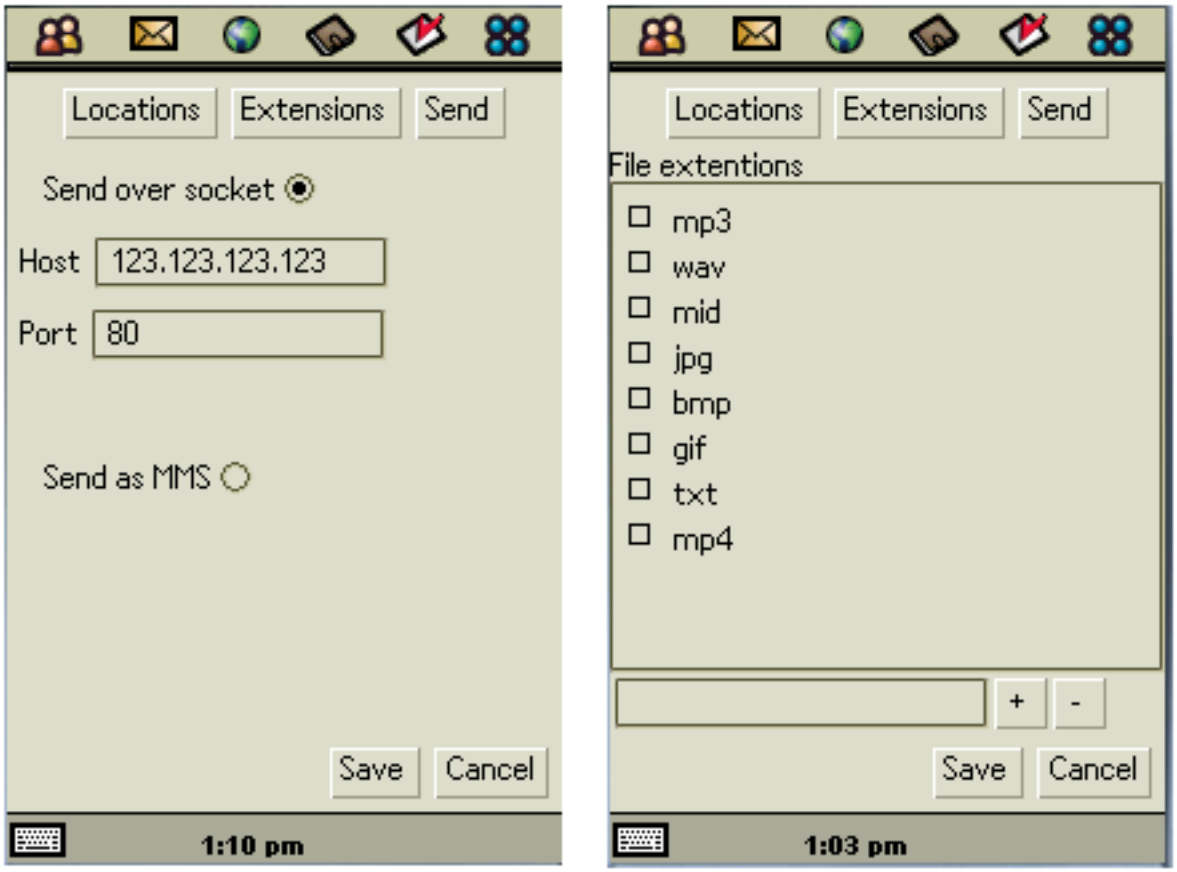

Figure 5.2. Screenshot of the BTDT setup

By clicking the Send button on the top, the user is given the screen in shown on the right in Figure 5.2 [34]. Here the user can specify the address of the server, and the number that will be used in the from field of the message. 'Send as MMS' is not implemented, and will be ignored.

5.1.2. Where

Where is implemented, and is able to at least read a single position from the GPS device and store it as a GPSmll file. It is however somewhat unstable, and the user interface needs more work. Reading tracks has not been tested as of the time of writing.

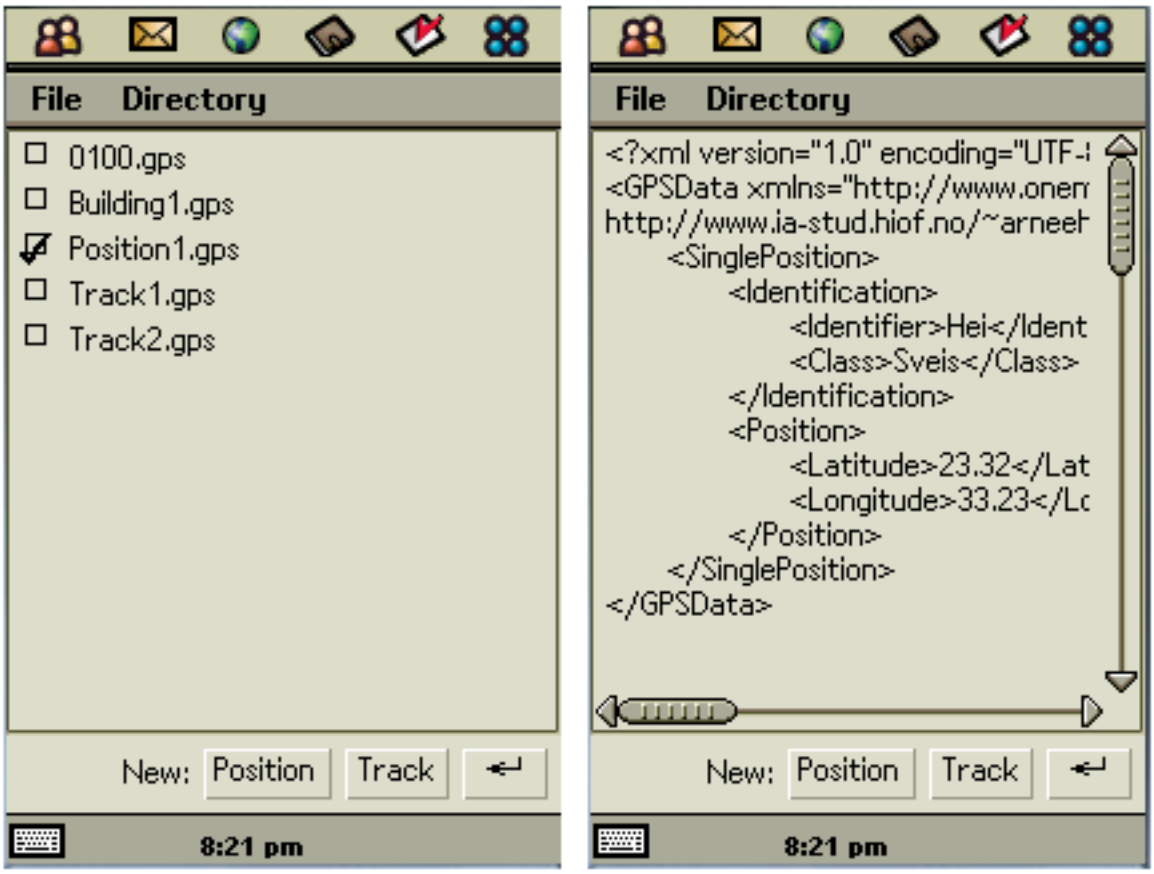

Figure 5.3. Screenshot of the Where list view (left) and text view (right)

Figure 5.3 [35] shows the list view . This is the view presented when starting the application. The menu named directory is used to select a location. The files in this location will show up in the list. New files will also be saved to this location. When selecting a file this file will be presented in the text view (see Figure 5.3 [35]).

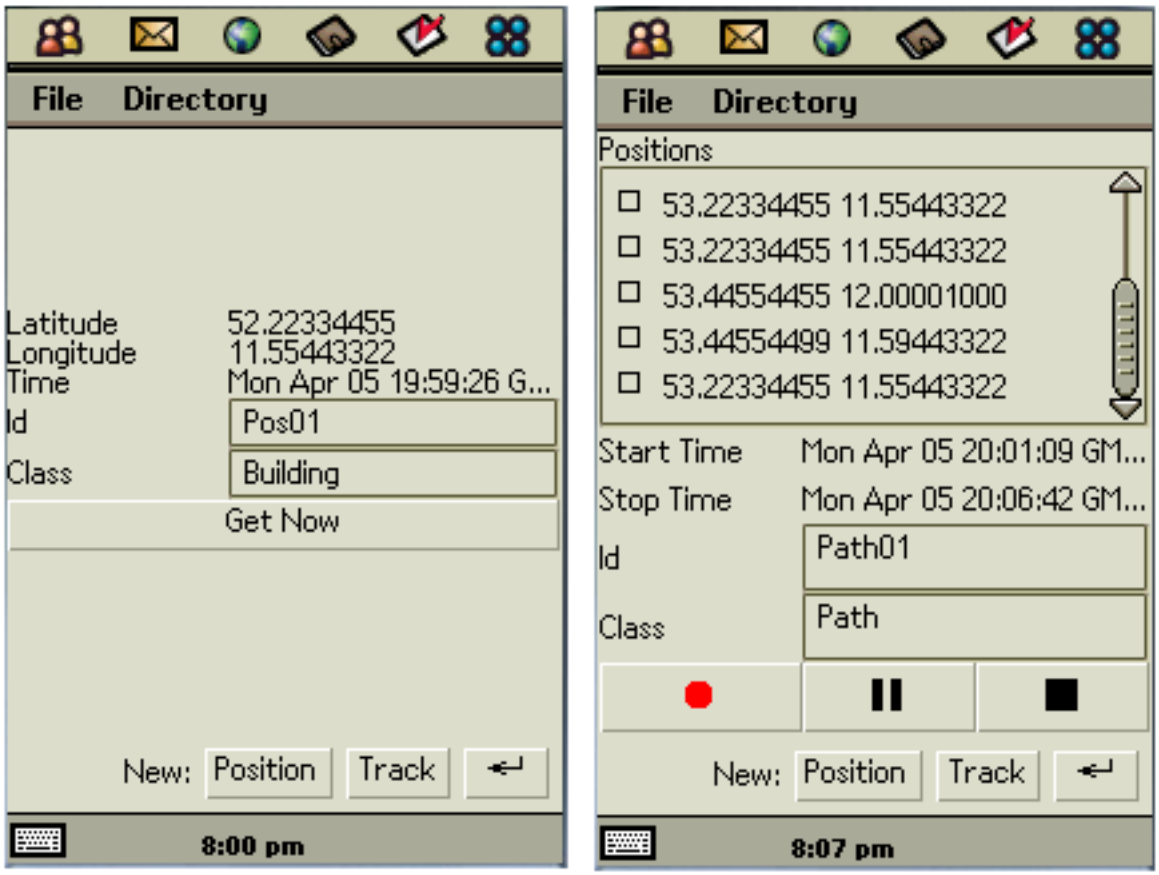

Figure 5.4. Screenshot of the Record Position View(left) and the Record Track View(right)

To create a new track or position the user taps the new position or track button. The Position View (Figure 5.4 [36]) contains only one button, which will cause the application to read from the GPS and display the result. The Track View shown in Figure 5.4 [36] has three buttons. These are record, pause and stop. If paused the track can be continued later even if the application is deactivated, but if stopped the track will be written to file when if deactivated. If still recording the track will be discarded. Two text fields allow the user to write an Identifier and Class for the track or position.

The rightmost button on the row of buttons is the return button. This will take the user back to the list view. When returning from the position or track view, the received data (if any) will be written to file. The filename will be made using the Id field or the current time if Id is not specified.

5.2. Further Improvements and Additions

The software developed in this project is usable on it's own. On the other hand there is great potential for improving it. Some changes is needed for the application to become more stable, others to make it more useable.

5.2.1. Known Bugs

ListView in Where. The folder menu created by the ListView widget includes a selection 'All'. This does not function as it is supposed, and may even cause data to be saved outside of the user-visible part of the file system. To ensure that errors don't occur this menu item has been disabled.

Setup in BTDT. BTDT stores it's setup and status using a BTDTSetup object. If the BTDT class is modified the setup file can't be read, and all configuration is lost. This only occurs when upgrading, and the class in question is modified. So this is not likely to be a problem for end users.

5.2.2. Stability Issues

Some errors may be more significant, and may even cause the application to malfunction or terminate untimely.

Error checking when using GPSLib. As mentioned earlier the error checking in the GPS library is weak and somtimes missing. The native implementation had some bugs that prevented it from returning error values, but this was easily fixed. The Java classes however absorb the error values before it reaches the application. I also implemented an utility class responsible for using the GPSLib classes, hence there are 3 layers of code between the native implementation and my application. This makes it hard to follow what happens.

When I started programming the GPS part I was not sufficiently aware of how the GPS library worked to predict this. The result is that my application is not able to handle errors in an optimal way. The GPS part of the application should be rewritten, not using all the GPSlib utility classes but rather work directly with the native implementation. While maintaining the interface to other parts of the application.

The current implementation is able to handle errors without the application crashing. I have not been able to retry connection after a connection failed though, or reconnect after a disconnect. The native implementation starts a thread on connect, and I beleive the problem is that this thread is not terminated properly on disconnect.

Safer XML. As mentioned there is currently no real validation of XML documents against the GPSmll schema. This is not an issue for the client, as there is no functionality that needs to use the documents. The server however needs documents to be useable therefore it could be desirable to verify the XML documents generated.

Splitting messages. When the MMS is sent all new media is added to a single message. It may be desireable to send the message in several smaller pieces. If the data is to be sent as ordinary MMS (as opposed to over a socket connection), this would be a requirement as there often is a maximum size allowed by the operator.

5.2.3. Improvements of Functionality and User-friendlyness

Although the application is usable in its current state, there are some changes needed for it to be adopted by ordinary users. Feedback to the user is scarce, and often cryptic for non-programmers. This is because most of the feedback given is through the TextArea component used for debugging. Better feedback would significantly increase the user-friendlyness of the application.

Handling network errors. Networking exceptions are handled in the current version. Quality feedback to the user is however missing. If a connection can't be made the user should be informed of why, to aid her in rectifying the error. Also the user should be given the possibility of storing a message for sending later if a network error occurs.

Hide the filesystem completely. The setup dialog for BTDT currently require the user to enter the full path of locations to search for media. This is not very user-friendly, and violates the style guidelines. This layout of the dialog have been useful during development, as it is closely related to how the application works. In a final product it should be removed, or at least changed. It is probably useful for users to be able to filter what types of media to include in the MMS, but this should be presented in a simpler manner.

More meta information in the message (Get the Message?). The current format of the message is very simplistic. The client adds whatever data it finds, and its left up to the serer to interpret the content. The data could be made more useful by adding some information about the content. In the current solution the only clue given to the server is the MIME type of each media object, and if this is unknown to the server it will not even

know what type of data its dealing with. Neighter is the original filename saved, but reconstructed from the MIME type. Each media object has an identifier that could be used to refer to the media object in a meta document. This document could be set as the message's root element, which is the document describing the rest of the message. For this purpose we could use SMIL, which is the standard MMS root element, or define a format for meta information ourselves. The last approach would certainly grant most flexibility.

Intergrate message composition. When creating ordinary MMS messages, users are given some control of how the media objects are displayed. A certain piece of text or audio may accompany a certain picture and so on. In BTDT there is no composition, media is added in the sequence they are identified. Giving the user this possibility would greatly increase the efficiency of communication using this type of message, but the penalty comes in the form of added complexity. It has been an important goal to keep the application easy to use. This would also require the user to know two different ways to compose MMS messages, unless our implementation completely replaced the standard one.

This issue is closely related to the previous section on meta data. For adding both meta data and composition we could define as schema that mix SMIL, a format for describing the content, and maybe event GPSmll. A possible format of such a document can be seen in Figure 5.5 [38] (note that this is not a format developed, just an idea of how to intergrate more information in a message).

```
<?xml version="1.0" encoding="UTF-8"?>
<LBMMessage>
    <Contents>
        <!-- A list of the media objects and their types -->
        <file type="audio" id="1">err_desc.wav</file>
        <file type="image" id="2">text1.jpg</file>
    </Contents>

    <Composition>
        <!-- SMIL markup here -->
    </Composition>

    <MetaData>
        <Comment relation="1">
            Voice note describing the fault seen in picture <Reference ref="2"/>.
        </Comment>

        <Location relation="2">
            <!-- GPSMll markup here -->
        </Location>
        <Comment relation="2">Picture showing the fault.</Comment>

    </MetaData>
</LBMMessage>
```

Figure 5.5. A Possible Instance of A Flexible and Informative Message Format

Using ordinary MMS. Using ordinary MMS as a content carrier would open, at least parts of, the application to other platforms. This would however involve a lot of work implementing the server, or we would have to use proprietary software on the server.

Tie each media object to a position. A position is created independently from other media. When a message is sent a position is added to the message if one has been obtained in the recording period. Hence there is no guarantee that a position is sent. Also several positions may be sent. It is therefore up to the server or reciever to interpret the position. A single position could be associated with the message as a whole. As discussed above more flexibility can be achieved at the cost of added complexity.

5.3. Installing and Running

Instructions for installing, setting up GPRS with Telenor, running BTDT.

Chapter 6. Conclusions

Here I will sum up the previous chapters, and highlight the most interesting results. I will also discuss the solution in a commercial context.

6.1. Project Description Redux

In this project I've aimed to implement a client for creating and sending location bound media on a Sony Ericsson P800/P900 mobile phone. Location bound media is media objects, such as video, audio and images, tagged with the exact location where the object was created.

The application developed was intended to be easy to use, and seamlessly integrate with the Symbian environment. Media is created using the standard tools already installed on the phone. For data transfer we wanted to use MMS.

Comparing what has been achieved with the original project description text, the application is not as much a OneMap client, but more of a service that uses OneMap as one of two data providers, with the client part, described in this report, being the other.

6.2. The Implementation

The P800/P900 has proven to be an ideal platform for the service implemented. Using PersonalJava makes it easy to get started developing, and JNI give access to more low-level functionality if needed. The phone comes with applications for creating media using the camera, audio recorder etc. It has a fairly fast processor and network connection.

Though not completely market-ready the application developed does what we intended, and we have been able to identify media and send it to the server over the GPRS connection.

6.3. Commercial Interest

Is the application developed interesting in a commercial context. What additions are needed.

6.4. Experience From Working With This Project

Experience from working with the project, not the implementation. Better planning? Recourses used.

Bibliography

[1] C. Turfus, Using MMS on Symbian OS phones - Parts 1 [http://www.symbian.com/developer/techlib/papers/mms/mms.pdf], 2 [http://www.symbian.com/developer/techlib/papers/mms/mms_part2.pdf] and 3 [http://www.symbian.com/developer/techlib/papers/mms/mms_part3.pdf] , Symbian DevNet

[2] J. Pagonis and J. Dixon, Location Awareness and Location Based Services - Part 1: Positioning and Terminology [http://www.symbian.com/developer/techlib/papers/messaging/LocalAwareness_LBS_01.pdf] , Symbian Ltd.

[3] Project OneMap [http://www.onemap.org]

[4] Faculty of Computer Sciences, Østfold University College [http://www.hiof.no/index.php?ID=3&lang=eng]

[5] G. Misund, Digital Maps [http://www.ia.hiof.no/digmap/pages/Prosjekter.html]

[6] G. Misund, Digital Maps Projects [http://www.ia.hiof.no/digmap]

[7] M. Brain and T. Harris, How GPS works [http://electronics.howstuffworks.com/gps.htm] , How Stuff Works

[8] UIQ Style Guidelines [http://www.symbian.com/developer/techlib/papers/uiq/uiqstyleguide/] , Symbian Developer Library

[9] Visual Basic on symbian [http://www.symbian.com/developer/development/vb.html] , Symbian Developer

[10] Java 2 Micro Edition [http://java.sun.com/j2me/] , Sun Microsystems Inc.

[11] A. Newman, Differences between PersonalJava and MIDP Java Environments [http://www.symbian.com/developer/techlib/papers/PJAE_MIDP/PJAE_MIDP_2.pdf] , Symbian Ltd.

[12] MIDP [http://java.sun.com/products/midp/] , Java Microsystems Inc.

[13] UIQ Application Development Tutorial [http://www.symbian.com/developer/techlib/tutorials/uiqappdev/tutorial/index.html] , Symbian Developer Library

[14] Designing for UIQ [http://www.symbian.com/developer/techlib/papers/uiq/symbian_designing_for_UIQ.pdf] , Symbian Developer Library

[15] I. Hutton, Implementing Applications the UIQ way [http://www.symbian.com/developer/techlib/papers/Qapps/ianh.html] , Symbian Developer Library

[16] M. Kotsbak, SonyEricsson P800 - GPS [http://symbianos.org/cgi-bin/viewcvs.cgi/P800GPS/homepage/P800GPS.html?rev=1.12]

[17] A. E. Hansen, GPSLib code for P800 [http://www.ia-stud.hiof.no/~arneehan/digmap/gps.html]

[18] GPS/Location Markup Language (GPSml) [http://www.chaeron.com/gps.html#GPSml] , Chaeron Corporation

[19] J. Pagonis, GPRS Facts for the Internet Application Developer [http://www.symbian.com/developer/techlib/papers/GPRS/GPRSFactsInternetApplicationDeveloper_pt1.pdf], Symbian Developer Library

[20] Mobile Location Services, Nokia White Paper
[http://www.nokia.com/downloads/aboutnokia/press/pdf/mlbs.pdf] , Nokia Mobile Phones

[21] R. Lake, GML 2.0: Enabling the Geo-spatial Web
[http://www.galdosinc.com/files/GML2-EnablingTheGeoSpatialWeb.pdf] , Galdos Systems Inc.

[22] S. Cox, P. Daisey, R. Lake, C. Portele and A. Whiteside, OpenGIS® Geography Markup Language (GML) Implementation Specification [http://www.opengis.org/docs/02-023r4.pdf] , Open GIS Consortium, Inc.

[23] P. A. Vretanos, Web Feature Service Implementation Specification
[http://www.opengis.org/docs/02-058.pdf] , Open GIS Consortium Inc.

[24] J. Ferraiolo, F. Jun and D. Jackson, Scalable Vector Graphics (SVG) 1.1 Specification
[http://www.w3.org/TR/SVG/REC-SVG11-20030114.pdf] , W3C®

[25] J. McGeough, Wireless Location Positioning based on Signal Propagation Data
[http://www.wirelessdevnet.com/library/geomode1.pdf] , Digital Earth Systems, Inc

[26] Cel-Loc Location Technologies Inc. [http://www.cell-loc.com/main.html]

[27] SnapTrack Inc. [http://www.snaptrack.com/index.jspl]

[28] PersonalJava [http://java.sun.com/products/personaljava/] , Sun Microsystems, Inc.

[29] Nokia Developer's Suite for MMS Version 1.1 [http://forum.nokia.com/main/0,6566,034-14,00.html]

[30] XML Light [http://www.softcorporation.com/products/xmllight/] , SoftCorporation LLC.

[31] Open GIS Consortium, Inc. [http://www.opengis.org/]