NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET

FAKULTET FOR INFORMASJONSTEKNOLOGI, MATEMATIKK OG ELEKTROTEKNIKK

NTNU

HOVEDOPPGAVE

Kandidatenes navn: Sigurd Gimre og Hege Servold

Fag: Datateknikk

Oppgavens tittel (norsk): Generisk rammeverk for utvikling av lokasjonsfølsomme applikasjoner

Oppgavens tittel (engelsk): Generic framework for development of location-aware applications

Oppgavens tekst:

Denne oppgaven går ut på å spesifisere, utvikle og teste et generelt rammeverk for lokasjonsbaserte systemer for turistguiding på mobilt utstyr (mobiltelefon og PDA). Første del av oppgaven går ut på å spesifisere arkitektur og modeller for systemet, samt implementere deler av dette. Den andre delen av oppgaven er å utvikle en demonstrator som er tilpasset en spesiell attraksjon. Prosjektet er et samarbeid mellom MOWAHS, Telenor og Klipp og Lim Media.

Oppgaven gitt:	20. januar 2004
Besvarelsen leveres innen:	15. juni 2004
Besvarelsen levert:	11. juni 2004
Utført ved:	Institutt for datateknikk og informasjonsvitenskap
Veiledere:	Alf Inge Wang og Steinar Brede

Trondheim, 11. juni 2004

Alf Inge Wang
Faglærer

Abstract

Today, several tourist attractions use handheld devices that act as tour guides to give the customers an improved experience and better knowledge of the attraction. A graphical user interface on the devices provides the users with information through sound, pictures and video. In order to improve the information delivered to the users, some of these guides are location-aware. However, location-aware tour guides are expensive to develop. They have to be developed tailored for each attraction, which is a time consuming job.

By having a framework for the development of location-aware tour guides, the tour guides will be both easier and less expensive to develop. In addition, it will be easy to make changes to the guides if necessary.

This project resulted in the development of a prototype for a framework for location-aware tour guides. The framework consists of three main tools; a *Creator Tool*, a *Statistical Tool* and a *Runtime System*. The *Creator Tool* is used to create and configure new tour guides. The *Statistical Tool* is used by the staff of the attraction. It generates statistics based on information stored in a log. The *Runtime System* is the system that provides the mobile devices information, adjusted to their location during a guided tour. To demonstrate the use of the framework, we have developed two client applications, one for PDAs and the other for cellphones. Both applications are electronically location-aware tour guides made for the Nidaros Cathedral.

The cellphone application is an innovative application that may result in a tremendous evolvement in the development of cellphone applications. It shows great potentials in the area of location-aware cellphone applications with high accuracy, and can be used not only in tour attractions, but also in several other fields. Thus, the project has gained a great deal of publicity, both from newspapers and television.

Preface

This report contains the documentation of the master thesis in the course TDT 4900 ”Hovedoppgave Sivilingeniørstudiet” by Hege Servold and Sigurd Gimre. The title of the thesis is ”Generic framework for development of location-aware applications” and it is a part of the Master of Science degree at the Norwegian University of Science and Technology. The thesis is a continuation of the work performed in the course TDT 4735 ”Systemutvikling fordypning”.

This report has been written under the supervision of PhD Alf Inge Wang. The report is a part of the MOWAHS project, which is supported by the Norwegian Research Council. The development of the thesis has been carried out at Telenor Research and Development’s offices at Tyholt in Trondheim.

We would like to thank our supervisors Alf Inge Wang from NTNU and Steinar Brede from Telenor. You have given us invaluable council during the 20 weeks we have been working with our thesis. We have appreciated all feedback we have received. We would also like to thank Per-Christian Bjelke and Terje Rafelsen from Telenor, and Ottar Dahle and Tom-Kristian Bardal from Radionor. You have all helped us with the implementation of the framework. In addition, we would like to thank employees from Klipp og Lim Media for their contribution regarding the Nidaros Cathedral tour guide. Finally, we would like to thank Skandia Take Off for a scholarship to the thesis.

Trondheim, June 11th, 2004

Hege Servold Sigurd Gimre

Contents

List of Tables

List of Figures

Part I

Introduction

The first part of the project presents the background and foundation of this master thesis. It gives an introduction and overview of the project that is accomplished. Part 1 also presents the structure of the report, and gives a presentation of the information enclosed in the CD.

Chapter 1

Introduction

This chapter provides a brief introduction to the project report and contains a description of the motivation, context, project description, process and method. The last section gives a structural presentation of how this document is organized.

1.1 Motivation

Today, several tourist attractions use handheld devices that act as tour guides to give the customers an improved experience and better knowledge of the attraction. A graphical user interface on the devices provides the users with information through sound, pictures and video. To improve the information delivered to the users, some of these guides are location-aware. Location-aware applications can be defined as *applications or services in which the location of a person or an object is used to shape or focus the application or service* [DCMC01]. A drawback with current location-aware tour guides is that they are expensive to develop. Hence, it is often too expensive for the various attractions to invest in location-aware tour guides. In addition the tour guides have to be tailored for each attraction, which is a time consuming job. Another problem is that it is difficult for the attraction staff to make changes to the guides.

A solution to this problem is to make a framework for developing location-aware tour guides. A framework can be defined as *an application ”skeleton”, which captures the essential features of a class of applications and can be ”easily” instantiated to produce (some aspects of) a specific application in the class* [CP03]. A framework will make it easier to develop location-aware tour guides. In addition the costs of the development will be reduced. These factors make it easier for tourist attractions to invest in location-aware tour guides.

The master thesis aims to develop a framework for location-aware applications.

The thesis is a continuance of our fall project ”Lokasjonsbasert guiding” that was completed in cooperation with Lars Klemetsaune. The fall project investigated, developed and tested a location-aware prototype for guiding on PDAs. The prototype was tailored to the Nidaros Cathedral. For a total overview of the project see [GS03].

The fall project and the master thesis are independent of each other, but experiences from the fall project are reused in this thesis.

1.2 Project description

The final project description is developed by the authors of this master thesis, and approved by the teaching supervisors and Klipp og Lim Media.

Generic framework for development of location-aware applications
This thesis is going to specify, develop and test a generic framework for development of location-aware applications for tour guiding on handheld devices (cellphone and PDA). The first part of the thesis is to specify architecture and models of the system, and implement parts of this system. The second part of the thesis is to develop a demonstrator that is tailored to a specific attraction.

The project is cooperation between Mobile Work Across Heterogeneous Systems (MOWAHS), Telenor Research and Development (Telenor R&D) and Klipp og Lim Media.

1.3 Project context

The work in this project has been carried out as a part of the research project MOWAHS [CN]. MOWAHS is a project supported by the Norwegian Research Council in its IKT-2010 program and is carried out jointly by the Department of Computer and Information Science's groups for software engineering and database technology at the Norwegian University of Science and Technology (NTNU). The project has two parts: process support for mobile users using heterogeneous devices (PC, PDA, mobile phones) and support for cooperating transactions/workspaces holding work documents. Although our framework is intended for tour guides, it can also be extended to be used to provide relevant information for mobile workers related to their location. This usage is of high importance for the MOWAHS-project.

The development of this thesis has been carried out at Telenor R&D's offices at Tyholt in Trondheim. Telenor R&D has contributed both with technical equipment and technical knowledge. The thesis was a part of the Arena for Research on advanced Telecom Services (ARTS) Project [Tel] at Telenor. The ARTS project is collaboration between NTNU, Telenor R&D, Ericsson and Abelia Innovasjon, and is also supported by Norwegian Research Council. The objective of the project is to heighten the national competence level related to advanced telecom services.

The thesis is performed in collaboration with Telenor R&D; by request from the Trondheim company Klipp og Lim Media.

1.4 Project process and method

A goal for the development of this project was to work systematic and structured to effectively make the most out of the time we had in disposal. Therefore, the project has followed a predefined process called iterative waterfall model, which is based on the waterfall model.

1.4.1 Waterfall model

The waterfall model inspires the software engineering process performed in this project. The classic waterfall model consists of five phases [GD98]:

1. **Analysis phase**
   The goal of this phase is to find out exactly what is going to be solved. The requirements of the project are determined. It is important to understand what the inputs and the outputs to the system are.

2. **Design phase**
   This phase determines possible solutions to the problem, and also alternative solutions. The goal is to refine a design that can easily be converted to code.

3. **Implementation phase**
   This phase converts the selected design into computer programming language and other chosen technologies.

4. **Testing phase**
   The goal of the testing phase is to verify that a program or solution works correctly and fulfills the requirements of the program.

5. **Release / maintenance**
   After the testing is finished, the software is released. This means that the software is put to work in a real working environment. After a release, bugs often appear. This generates a need for maintenance.

Figure 1.1 visualizes the waterfall model.

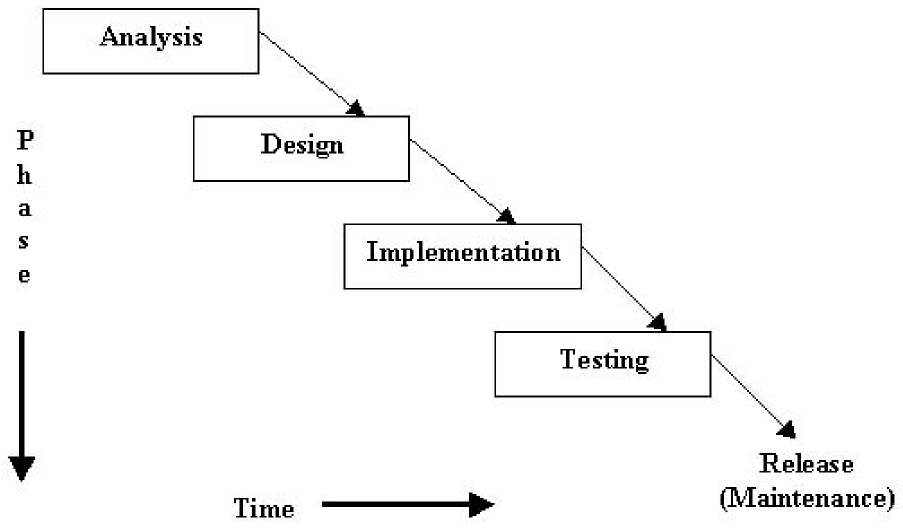


Figure 1.1: Waterfall model

The waterfall model requires one phase to be completed before the next phase begins. This is difficult to accomplish in practice. It is often important to be able to return to the previous phase even though it has ended. Thus, the iterative waterfall model is required.

1.4.2 Iterative waterfall model

The iterative waterfall model allows one to go back and forward in the phases of the model if necessary. Thus, it is not critical that one phase is completed before you move on to the next phase. Figure 1.2 visualizes the iterative waterfall model.

1.4.3 Way of working

Based on the project members' earlier experiences with engineering processes and due to the complexity of the project, we have chosen to follow the iterative waterfall model. When the project started, a *Gannt* diagram that defined the phases and important milestones was made. The diagram can be found in Appendix D.

During the execution of the project, effort has been made in order to follow the milestones. However, the iterative waterfall model has allowed us to move back and forward between the phases, making the accomplishment of the project go smoothly.

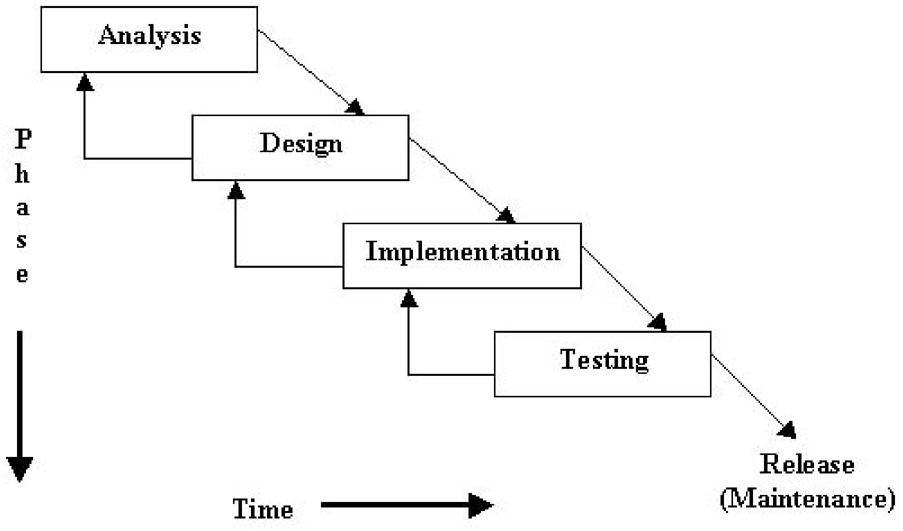


Figure 1.2: Iterative waterfall model

1.5 About the report and the accompanying CD

This section describes how to read and use the report.

1.5.1 Report outline

The report consists of 6 main parts, which contain a logical separated portion of the project.

Part 1: Introduction

This part contains a description of the motivation and foundation of the master thesis. It also provides a description of the software engineering process used in the project.

Part 2: Prestudy

This part provides general information of topics related to the development of location-aware applications.

Part 3: Construction and Implementation

This part presents all system specific documentation of the *Generic framework for development of location-aware applications*. An important contribution is chapter 9 presenting the system's architecture.

Part 4: Demonstrator

This part provides an overview of the two applications developed for the demonstration of the framework in the Nidaros Cathedral.

Part 5: Discussion, conclusion and future work

This part sums up and concludes the results of the project.

Part 6: Appendix

The Appendix consists of code documentation with examples, the press release and two articles published in Teknisk Ukeblad and Verdens Gang.

1.5.2 CD content

A CD is included with the report. This CD contains files that are created during the project period. It is a complement to the report, and will be helpful for future work with the project.

The CD contains the following folders:

- **creatorTool**
  Contains source and project files necessary to run the Creator Tool and also the driver used to connect to MySQL.
- **PDA_application**
  Contains compiled Flash files, music and a PDA standalone Flash player.
- **report**
  Contains the report as a pdf document.
- **runtimeSystem**
  Contains source code, a compiled lib file and javadoc for the Runtime System.
- **reportages**
  Contains video clips covering the system.

Part II

Prestudy

In the beginning of the project period, a prestudy was carried out. The focus was studying different positioning technologies, performing a state of the art study, get an overview of other framework systems and get familiar with different coordinate systems. The goal of the prestudy was to stimulate the creativity and bring motivation to the forthcoming construction and implementation.

Chapter 2

Location identification systems

The identification of a device's position can be performed with different techniques. Each technique has its strength and weaknesses. The position is given at various levels of granularity that differ from a few centimeters up to hundreds of meters.

This chapter provides an overview of some of these location identification systems and a summery discussing how to best use the strength of each system to a full extent.

2.1 GPS

The Global Positioning System (GPS) is one of the most known positioning systems [Gar04]. It is a satellite-based navigation system made up of a network of 24 satellites (Figure 2.1) placed into orbit by the U.S. Department of Defense. GPS was originally intended for military applications, but in the 1980s, the government made the system available for civilian use. GPS works in any weather conditions, anywhere in the world.

Figure 2.1: GPS - satellites

Today GPS receivers have accuracy within 15 meters. When using Differential GPS (DGPS) the signal will be corrected within an average of three to five meters. DGPS consists of a network of towers that receive GPS signals and transmit a corrected signal by beacon transmitters. In order to get the corrected signal, users must have a differential beacon receiver and beacon antenna in addition to their GPS.

A drawback with GPS is that it is not suitable in urbane centers. In these areas it will meet too much noise interfering with the signals. To use GPS you need free line-of-sight which means that it cannot be used inside buildings. Another drawback with GPS is that mobile terminals need an extra GPS receiver chip to find their position.

An example of a system that benefit from GPS is Digital Angel. The system uses GPS satellites to locate patients [Bon04], and it monitors a patient's vital signs, such as temperature and pulse, through a special biosensor embedded in a wristwatch. The data is sent wirelessly to a pager device that has a GPS receiver, where the patient's current location will be measured. Both the location and the patient's condition will be sent wirelessly to a satellite and further stored in a database. Subscribers to the service will be able to log on from any Internet-connected device and read information about the patient.

2.2 GSM-positioning

Global System for Mobil Communications (GSM) is the standard for mobile telecommunications used in Europe [MT02]. The first generation of cellular phones were analogue, but GSM has been designed from scratch as a fully digital system without any compromises concerning backward compatibility. In Norway, the GSM system covers all urban areas and most roads. However, there is typically no coverage in the mountain and the countryside, except a few of the most popular places.

The mobile communication system has a cell based structure, illustrated in Figure 2.2. Thus the geographical area that is covered by the network is divided into a number of smaller areas called cells. The cells have a radius from 50 meter to 15 kilometer [Mol03]. At the centre of each cell is an access point to which all the telephones in the cell transmit. Each cellular phone is logically located in one specific cell and under the control of that cell's base station. In an area, for example in cities, where the number of users has grown to the point where the system is overloaded, cells are split into smaller cells.

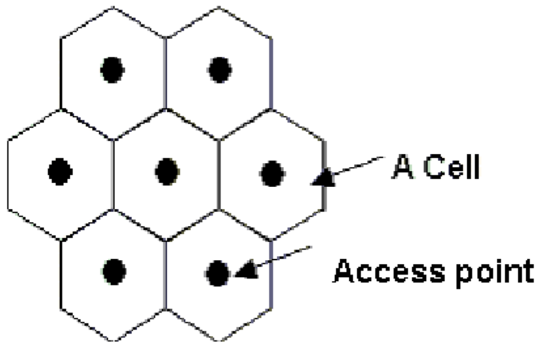


Figure 2.2: Cells in a GSM network

The positioning of the GSM device is based on which cell the user is located in. Thus you will have a more accurate position in urbane areas where the cells are small.

There are many commercial location-aware systems made for the GSM network. One examples is Buddy [Net04]. Buddy is a SMS-service for cellphones where you can make friend-lists. By using these list you can find the location of your friends. Another location-aware system based on the GSM network is hvor.no [Geo04a]. hvor.no uses SMS or MMS to find for example the nearest drugstore or gas station. GSM-based location systems are easily charged, which have made them a great success.

2.3 WLAN-positioning

WLAN technology[1] allows devices to immediately connect to Internet/Intranet in a range of 100 meters without any cables using an access point and a PCMCIA[2] wireless card. As in a classic LAN, a group of devices with wireless cards installed creates a wireless Intranet. The latest WLAN standard can theoretically share files at 108 Mbps, however real throughput is closer to 50Mbps [Ita04][thg04].

The position of a device using WLAN can be determined by identifying the closest base station. The accuracy will be about the same as the base stations range. The Lan-

[1]Refers to WLAN technology as the IEEE 802.11x standards
[2]Personal Computer Memory Card International Association

caster tour-guide is an example of a system that uses WLAN base stations for location determination [DCME01]. It uses a Tablet PC to present different data about the city. When a user is approaching for example a castle and gets inside the WLAN zone, he will receive information about the specific castle. The Lancaster tour-guide is further described in section 3.1.

WLAN positioning with base stations can in many cases give a too inaccurate position. Lately, different systems have been released to improve this condition. Examples are Cordis RadioEye [Com04] and Ekahau [Eka04]. These systems use different technologies to calculate user's location with accuracy down to a few meters. Both systems are being tested by Telenor R&D in Trondheim and will be introduced in the following sections.

A problem related to the use of WLAN is the high cost in terms of battery power, which may be a huge problem for mobile users. Moreover, walls and iron objects can interfere with the signal, making the position incorrect [GS03].

2.3.1 The Cordis RadioEye

The Cordis RadioEye is a small box with advanced antenna technology and a Linux-server that can decide the physical coordinates to every WLAN-terminal that are active in its coverage area, illustrated in Figure 2.3 [Com04]. The sensors decode the MAC addresses of the network users terminals and determine their geographical position with a typical accuracy of 1-2 meters.

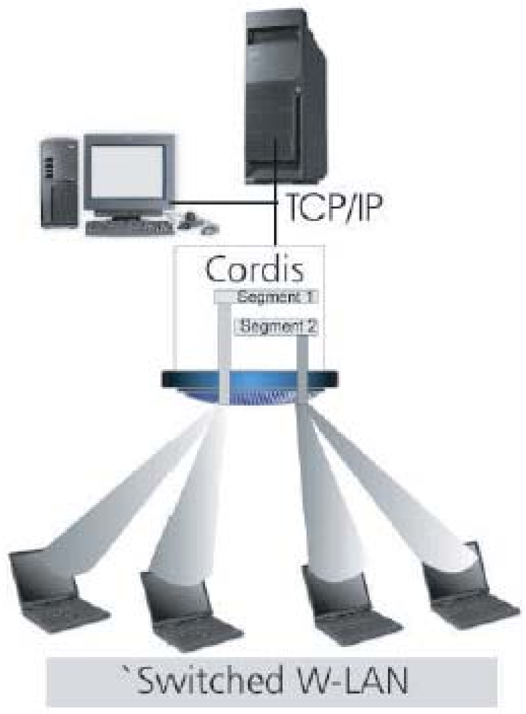


Figure 2.3: Cordis RadioEye System

In theory the Cordis technology may, in addition to position WLAN-systems, position every wireless system such as Bluetooth, GSM and UTMS. However, these systems are not yet developed.

Today, the Cordis RadioEye offer position in three different coordinate systems. These are Geodetic Latitude and Longitude, Universal Transverse Mercator and Rikets Triangulering 90. These systems are further explored in section 5. The coordinates are fetched by sending a XML-request to the Cordis RadioEye's web server.

Two elements influence the accuracy of Cordis RadioEye and limit its use. The first is the issue of line-of-sight. The more material between the mobile device and the RadioEye, the weaker location. Another limitation is the angle between the RadioEye and the wireless terminal. When the angle increases, the accuracy of the position decreases. At line-of-sight borderlines, and into some parts of the non-line of sight areas, the accuracy will be better than +/- 2 meter while outside much worse. Figure 2.4 shows the area covered by the Cordis RadioEye.

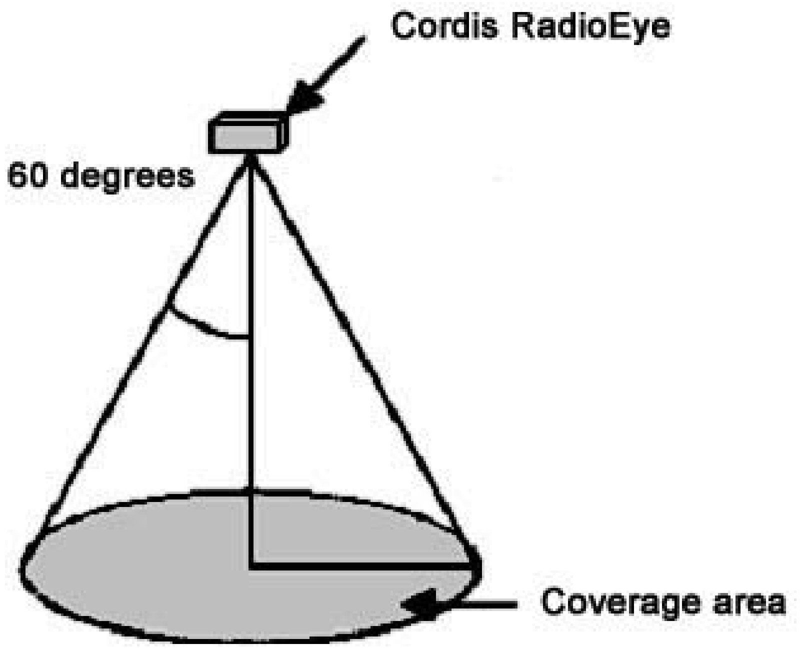


Figure 2.4: Area coverd by Cordis RadioEye

The Cordis RadioEye is primary for indoor use, but may also be used outdoor for micro positioning on an enclosed area. By using multiple units of Cordis RadioEyes, a larger area can be covered. 3D coordinates may be calculated by combining data from more than one RadioEye.

2.3.2 Ekahau

The Ekahau system is a set of software components that enables location tracking in any standard Wi-Fi network (802.11 a/b/g) [Eka04]. Unlike competing technologies that require special infrastructure on site, Ekahau is a pure software-only solution that works with any off-the-shelf Wi-Fi access point. Ekahau uses its own-patented Bayesian positioning algorithms to calculate the position of a WLAN device such as PDA, Laptop and Palm. More access points results in the better accuracy. 5-7 access points will provide up to 1-meter average accuracy and 3-5 access points provides up to 2-3 meter average accuracy.

Unlike many competing positioning technologies, such as Cordis RadioEye, Ekahau does not apply propagation or triangulation methods that suffer from radio wave multi-pathing, scattering or attenuation effects. Thus it is very suitable in environments with a lot of disrupting elements.

2.4 IrDA-positioning

Infrared Data Association (IrDA) is a standard that supports high data rates, from 2.4Mbps to 11.5Mbps [IrD04]. Since Infrared (IR) devices use infrared light, they depend on being in line-of-sight contact of each other as illustrated in Figure 2.5. This may be a problem in some positioning systems. In addition, the light pulses seldom transmits longer than 2 meters. Due to the short range and the need of free line-of-sight, great number of access points are needed. However, IrDA doesn't interfere with other electronic devices, which is a great advantage. It is also relative inexpensive compared to WLAN, introduced in section 2.3, and Bluetooth, introduced in section 2.5.

Figure 2.5: IrDA need direct line of sight

The Active Badge Systems (ABS) is an example of a system that benefits from IrDA [WHFG92][WH92]. It is a small computing device worn by personnel. Each badge has a globally unique code that is periodically broadcasted through an infrared interface. The infrared signals reflect off walls and furniture to flood the surrounding area, and are picked up by a network of sensors placed around the building. By determining which badges were seen by which sensors, it is possible to deduce the location of a badge. This further provides a hint to the location of the badge's owner. Applications in which Active Badge information has been used include telephone call routing, security and environmental control [WHFG92]. Another example that benefits from IrDA is the Cyberguide project [KP97]. It uses IrDA in the same way as the ABS for inside positioning, while GPS is used for outside positioning. The Cyberguide is futher described in section 3.2.

2.5 Bluetooth-positioning

Bluetooth is an open specification for short-range wireless speech and data communication [MT02]. Bluetooth uses radio waves, has usually a range of 10 meters and can share files at theoretically 1Mbps. There are also specifications where Bluetooth has a range up to 100 meters, though these are rarely used. Since Bluetooth is using radio waves there is no need for the line-of-sight between sender and receiver that IrDA has. Bluetooth can even communicate through walls of iron, which is a big advantage. Bluetooth also has two other advantages. It is cheap and its easy to use. The devices locate each other and communication is established without any user input at all [Fra04].

By using Bluetooth the position can be determined by identifying the closest base station, which works well because of Bluetooths short rang. Devices that are out of range will not be interrupted by the sender. There is a prototype tour-guide at the Museum in Grimstad which illustrates the use of Bluethooth for positioning [HT03]. This prototype provides the tourist different information depending on the base station it is connected to.

A problem using Bluetooth is the cost of setting up a new connection between 2 devices. It takes between 5 and 10 seconds [WJCK02]. If a user moves fast he can move through a zone of 10 meter in diameter in lesser time, and there will be no location available.

2.6 Ultrasound-positioning

Ultrasound is high frequency sound waves [Lip04][CMEE04]. Technically speaking a sound with a frequency above the audible range of human hearing, more than 20 000 Hz, is known as an Ultrasound. Ultrasound is best known for its medical use. Ultrasound or sonography, in medicine, is a technique that uses sound waves to study and treat hard-to-reach body areas. In scanning with Ultrasound, high-frequency sound waves are transmitted to the area of interest and the returning echoes recorded. Ultrasound is non-invasive, involves no radiation, and avoids the possible hazards such as bleeding, infection or reactions to chemicals.

Ultrasound has several desirable properties that makes it ideal for location-aware applications [tec04]. It does not penetrate solid walls, so location per room is simple to achieve. In addition, it does not require line-of-sight between the tags and the detector. Ultrasound waves are mechanical waves and are immune to interference. Ultrasound will not interfere with electronic equipment and is therefore ideal in environments that are sensitive to the electromagnetic radiation from radio systems and have needs for tracking equipment.

The Bat system from AT&T Laboratories in Cambridge is an example of a positioning system that benefits from Ultrasound [Lab04]. The ultrasonic location system is based on the principle of trilateration; position finding by measurement of distances. A short pulse of Ultrasound is emitted from a transmitter called a Bat attached to the object to be located, and the times-of-flight of the pulse to receivers mounted at known points on the ceiling is measured. The speed of sound in air is known, and the distances from the

Bat to each receiver are calculated based on this. Given three or more distances there are enough information to determine the 3D position of the Bat, and hence that of the object on which it is mounted. Figure 2.6 shows a user with a Bat.

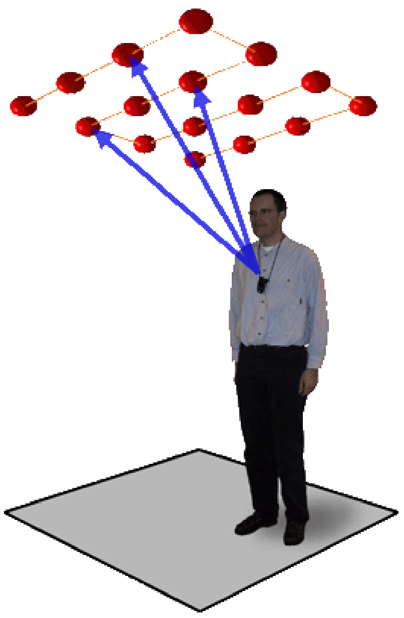


Figure 2.6: Bat system

A disadvantage with using Ultrasound for positioning is that the users needs to carry an extra device for transmitting signals that are not integrated in the mobile unit that is going to be located. Moreover, Ultrasound positioning is not suitable outdoor covering larger areas.

There is a Norwegian firm working with Ultrasound positioning systems. They have developed a system called Sonitor that are being tested at NTNU. This system is presented in the next section.

2.6.1 Sonitor

Sonitor is an indoor tracking system, which uses Ultrasound for deciding objects locations [tec04]. The Sonitor system uses a local ultrasonic communication network as shown in Figure 2.7. Equipment, files, records and people can quickly be located by signals transmitted from a small, attached electronic tag, similar to the Bat system in section 2.6. The tag continuously communicates its position to a computer-based detector network. The position of objects is tracked with an accuracy of 2-3 cm.

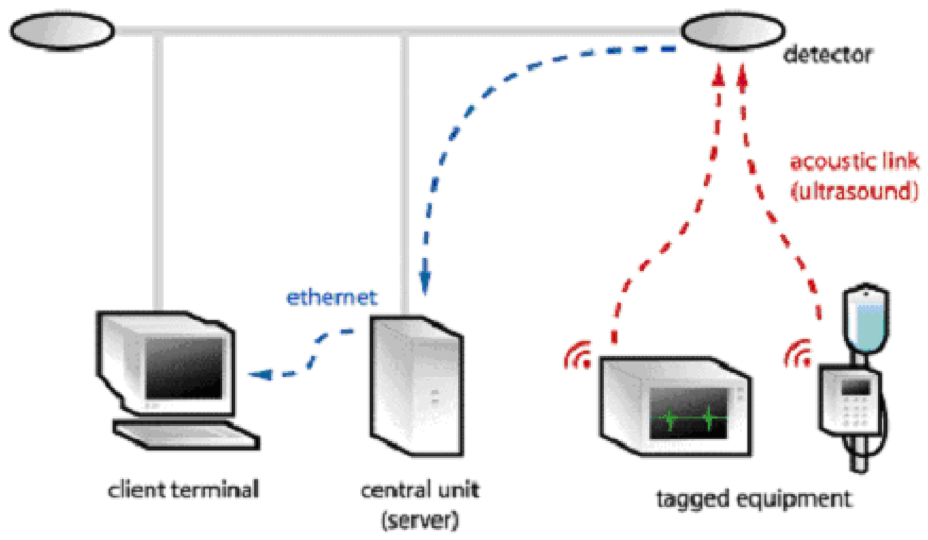


Figure 2.7: Sonitor network

Standard computer interface

The Sonitor system uses proprietary digital signal processing technology in the detectors in order to achieve a safe and reliable communication link. The location data is then stored on a standard database in the Sonitor server. The data can be accessed from multiple client terminals and located wherever convenient from the point of view of the institution. Another alternative is that an existing asset management system reads data from the Sonitor server's database and integrates it with the other asset tracking and reporting data of the users systems.

2.7 Summary - A swim lane of system

There exist many different sensing technologies for finding location. This section introduced the most familiar positioning systems, but only a few of the total number. Table 2.1 contains a summery of the positioning systems' properties. In addition to these systems there are also systems based on ground reaction force, physical contact, electrical contact and visible light among others.

Location-aware applications using one of these positioning technologies are numerous. Although there are many available applications, there are few commercial-killer-applications. One of the problems is that every sensing technique has its limitations. GPS have low accuracy in urban areas while GSM-positioning has low accuracy all places except urban areas. WLAN-positing use a lot of power, IrDA-positioning needs line-of-sight while Ultrasound positioning needs an extra device to work.

To be able to use the most appropriate technology under every circumstance, there is need for a system that can use them all. A solution is to use a location server layer that abstract the position determination from the end application. The end application should not have to worry about the position. It only needs to know that it receives the best position available.

In our framework for developing location-aware applications, a location server (LS) will be a central part of the architecture. The LS will do all the communication with the positioning systems. This feature will be a great advantage in the search for making excellent location-aware applications.

Pos. Tech.	**Positive**	**Negative**
GPS	Works anywhere in the world, high accuracy down to a few meters	Need line-of-sight to satellites, need extra chip to use
GMS	High accuracy in urban areas and easy to charge	Low accuracy outside urban areas, is charged
WLAN	Can use extra system to better the accuracy, don't need extra chip as with GPS, IrDA and Ultrasound	Low accuracy, down to about hundred meters, Signal can interfere with walls and objects, high battery use
IrDA	Does not interfere with other electronically devices, inexpensive compared to for instance WLAN	Need line-of-sight, short range (ca 2m), need many extra devices to use
Bluetooth	No line of sight needed, short range a plus for position determination	The cost of setting up connection can take 5-10 seconds
Ultrasound	Do not need line-of-sight, do not interfere with other electronically devices	Not suitable over larger areas, need an extra device

Table 2.1: Summery of positioning technique properties

Chapter 3

State of the art study

Chapter 2 presented different positioning techniques, the advantages and disadvantages to these. This chapter will introduce three specific location-aware tour guides and study each system and the technologies used.

3.1 Lancaster tour guide

The Lancaster tour guide application provides an electronic handheld guide that visitors to Lancaster City can use to access information about the city, create tailored tours of the city, and access interactive services [DCME01]. The system consists of a tablet-based PC equipped with 802.11 wireless network capabilities. Figure 3.1 presents a Fujitsu Teampad 7600 used with the system.

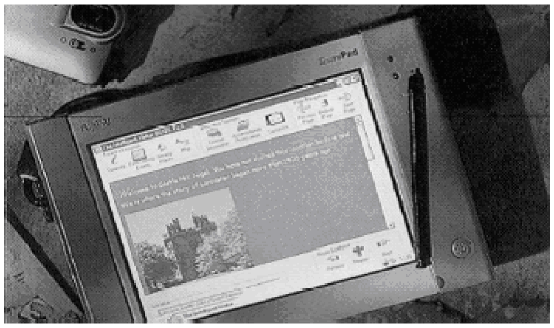


Figure 3.1: Fujitsu Teampad Table PC

The first version used GPS for locating the device, which was inexpensive and easy to use. However, GPS had some disadvantages. Most critical, to obtain an accurate positional fix, a GPS receiver must be able to locate at least three satellites, which precludes using the technology inside buildings and in areas where the satellite signal is blocked by buildings. Thus the developers decided to use WLAN positioning so that the system could be used inside buildings. They positioned the user by identifying the closest base station so the accuracy was about 50-100 meters, which was sufficient to display general information about the geographical area.

The Lancaster guide also offers custom guides based on user preferences. In the beginning, every user receives some questions and based on the answers they provide, a route is generated.

3.2 Cyberguide

At Georgia Institute of Technology there has been a project concerning mobile information called The Cyberguide project [KP97]. The project has designed several tour guide applications to investigate how the visitor at the GVU centre lab could be supported by location-aware services. The applications use both the current and former locations to give the correct information.

The system has been tested during the Universities monthly open houses. Visitors have, by using the Cyberguide system, been guided through the University receiving information about the places visited through for example applications on the MessagePad, shown in Figure 3.2.

Figure 3.2: Apple MessagePad with a GPS unit

The Cyberguide uses both indoor and outdoor positioning. IrDA is used inside, while GPS is used outside. LAN is used to transmit information. Through the network connection there is possibilities to send e-mail and read web pages. The Cyberguide system was made to work with different platforms and the application is tested on, among other, Apple MessagePad and Daulphin palmtop.

3.3 MUSE: Multi-channel web Framework for cultural Tourist Applications

The university of Bologna in Italy has developed a framework for developing Multi channel (MC) tourist guides called MUSE [CP03]. MUSE aims to offer an engaging education experience to communicate the meanings of history by use of location-based information.

The MUSE framework has been used to implement mobile and stationary systems for three MC applications for cultural guide; the first concerning the archaeological site of Pompei, the second concerning the Certosa di San Martino, the historical museum of Naples, and the third concerning the museum of Science history in Florentine.

The MUSE MC framework considers four different channels for presenting information to the users. These are the web channel, the on-site mobile channel, the on-site stationery channel, and the memory channel. The four channels will be presented in the following sections.

3.3.1 The web channel

The web channel is an Internet connected desktop of laptop. The user can navigate through web pages of the tourist application to learn more about it.

3.3.2 The on-site mobile channel

The on-site mobile channel is a hand-held device called Whyre, shown in Figure 3.3. Whyre is specifically designed for the MUSE project and uses location based information to guide users. Whyre has many different sensors including a GPS, a digital compass, gyroscopes, accelerometers, a camera and a WLAN based tracer.

Figure 3.3: Using Whyre, the MUSE on-site mobile channel

3.3.3 The on-site stationary channel

The on-site stationery channel is a stationary graphic station connected to a large display located at specific points of a cultural site. The display can give exact information about the area, or be used to show information from Whyre in a larger format.

3.3.4 The memory channel

The memory channel is a multimedia CD. During the on-site visit, the users can select any cultural material presented. They can also take digital pictures using Whyre and make albums. The final information can then be saved on a CD which the users bring back home as a memory.

3.4 Summary - Three tour guides

The state of the art study presents three different solutions on how location-aware application can be made and how the data are presented to the users.

The MUSE project has chosen to make a computer device called Whyre to present data.

The Lancaster tour guide uses a Tablet PC, while the Cyberguide uses different kinds of Pocket PCs.

Each device has its positive and negative aspects. To make a dedicated device as in MUSE would be preferable in many situations. However, there are many problems with this approach. The biggest problem is the expense, most Norwegian tourist attraction will not afford the needed investment. Similar to Whyre, Pocket PCs and Tablet PCs also have their limitations. Pocket PCs often have a very small screen, while Tablet PCs are too heavy to carry around.

It is important to always select the correct device at any time. In order to make the most suitable location-aware system for each distinct tourist attraction our framework will deliver all of the location data as XML. It will be up to the client application developers to choose which mobile device to use. Thus our system will be possible to use for each device that can parse XML.

Another feature worth considering when developing location-aware applications is the memory channels. In MUSE, the memories are stored on a CD that users can bring home. However, this is out of the scope of this master thesis. It is an issue for the people developing the client applications using our positioning framework.

Chapter 4

Location-aware frameworks

A framework is a system that can be used to generate specific applications within a specific topic, adjusted to the customers needs and preferences. It allows you to quickly and easy configure and deploy applications. These applications range from simple web sites to complex systems.

A framework can lead to several benefits for the end customer, amongst other:

- Time saving (shorter time to develop adjusted applications)
- Money saving (because the development takes less time)
- Easy to use, customers can do the job themselves (often wizards).

In the following sections, two different frameworks are presented.

4.1 The NEXUS Project

The NEXUS Project aims at the development of a generic infrastructure that serves as a basis for all kinds of location-aware applications [VS00]. The project includes both indoor and outdoor services.

The system is composed of four units that have to cooperate; the user interface, the sensor systems, the communication and the distributed data management as illustrated in Figure 4.1.

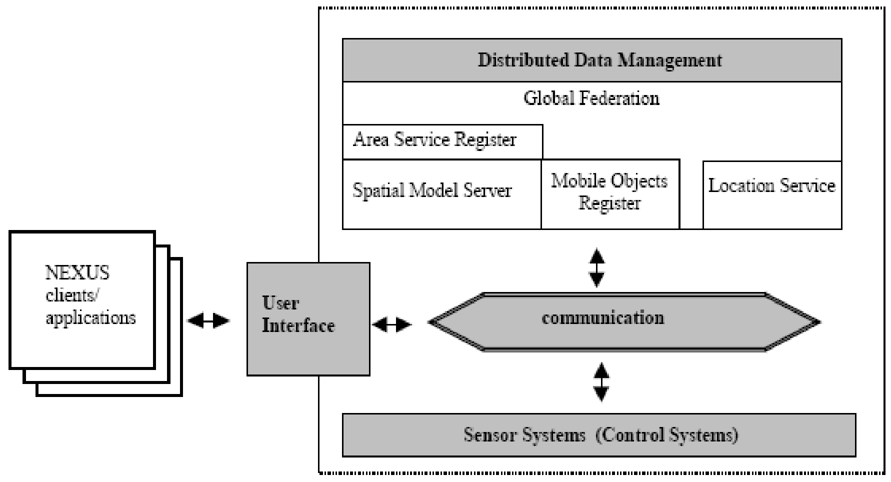


Figure 4.1: The architecture of NEXUS

The NEXUS clients access the platform via a standardized user interface which is running on the mobile device carried by the user. The interface has to adjust to the different kinds of clients, especially concerning the different level of computing power, different amounts of memory and different size of displays. The NEXUS system also has to guarantee platform independent usage.

The sensor systems have to measure both global indicators (like temperature) and object related information (like location). Therefore, the NEXUS station has to be equipped with appropriate sensors. The sensors record the different data and forward them to the data management component. The communication unit is responsible for the data exchange between the different components.

A central part of the NEXUS platform consists of the management of spatial models that represent the physical world. The spatial models have to manage data in multiple representations, and huge amounts of data must be stored and handled. Therefore, large and distributed databases have to be used.

Authorized users have the opportunity to change the properties of objects and propagate these changes to the appropriate server via the NEXUS station. For example, an authorized user could select a room in a building on the display of his computing device, change some attributes and send the new information back to the platform.

4.2 FLAME

The aim of the Framework for Location-Aware ModElling (FLAME) framework is to provide a configurable, generic software framework that reduces the effort required to develop location-aware applications to a minimum [CNS]. An additional goal of FLAME is to provide a framework that is highly configurable and extensible so that others can use it as a research platform. Some of the projects design goals are:

- Support for multiple sensor technologies
- A simple spatial model for the representation of locatable entities
- A simple event architecture for the presentation of location information to applications
- A queryable location database

The framework and its applications are largely event-driven in order to accommodate the real-time nature of the location information that they handle. The system is composed of four main units, illustrated in Figure 4.2.

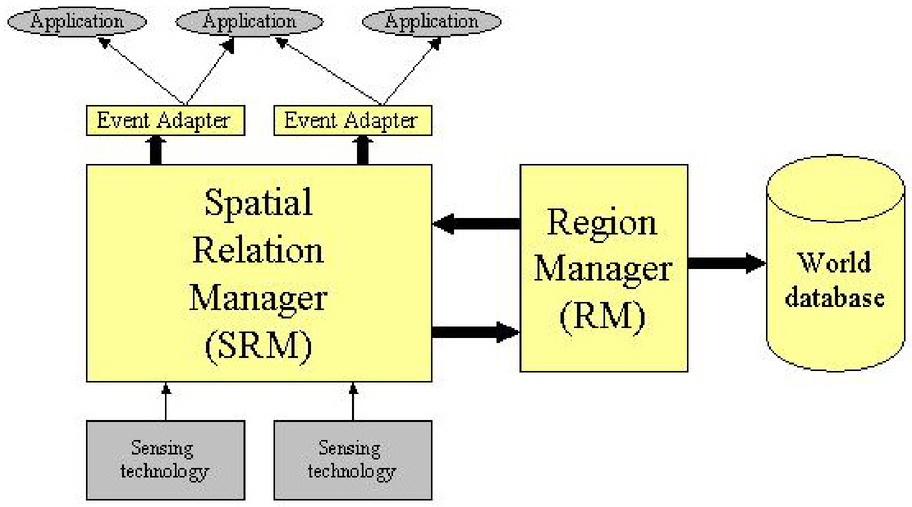


Figure 4.2: The architecture of FLAME

The database holds the initial states (like static regions), and it also holds a synopsis of the real-time location information. The region manager stores and retrieves regions from the database. The spatial relation manager generates application-related events in order to satisfy currently active subscriptions. The event adapters generate events, for instance a ”Person Movement Event” when it notices that a person has moved. Although events allow the transfer of location information from FLAME to applications, the system also allows non event-based applications to retrieve this information.

Several tools to facilitate the creation and management of location-aware applications are included in FLAME. A graphical World Builder tool allows users to quickly construct a model of their world without the need for programming. The SRMConfigurator tool is an application that can read a XML description of how FLAME should be configured, and it automatically creates and configures FLAME according this description.

FLAME also provides interface modules for several location technologies. A template is also provided which allows users to easily support other location technologies they may have.

4.3 Summary - Two location-aware frameworks

Chapter 4 presents two different frameworks for developing location-aware applications. Both frameworks are presented with their architectures in order to reuse the experiences other people have gained.

4.3.1 The NEXUS project

The NEXUS project has divided the sensors in a separate module. This idea will be used in the architecture of the framework we are going to develop. Another feature we will us is the standardized user interface between the clients and the rest of the system.

The NEXUS project uses a distributed database to store the necessary information. Since every attraction is not connected to the Internet, our framework will instead use a centralized database. The framework will be installed as separate systems at the various attractions with separate databases.

4.3.2 FLAME

FLAME uses a Region Manager to retrieve and store regions from the database. This idea will be used in our framework. In addition, FLAME consists of several tools, which will be used in our framework.

The FLAME system is event-driven. When an event appears, new information is sent to the applications. In our framework the client applications will initialize the events, thus this feature will not be used. The reason for this is to avoid applications on the clients that are listening on incoming events.

Chapter 5

Coordinate systems

Coordinate systems are used to specify locations on the surface of the earth. There are several different coordinate systems available that can be used when positioning mobile devices. Examples are World Geographic Reference System (GEOREF), National Grid Systems (NGS) and Universal Transverse Mercator (UTM). More about these and other coordinate systems can be found at [Dan04].

The following sections present some of the most used coordinate systems.

5.1 Geodetic Latitude and Longitude

The most commonly used coordinate system today is the latitude and longitude system. The Prime Meridian and the Equator are the reference planes used to define latitude and longitude. The geodetic latitude of a point is the angle from the equatorial plane to the vertical direction of a line normal to the reference ellipsoid. The geodetic longitude of a point is the angle between a reference plane and a plane passing through the point, both planes being perpendicular to the equatorial plane. Figure 5.1 illustrates this principal.

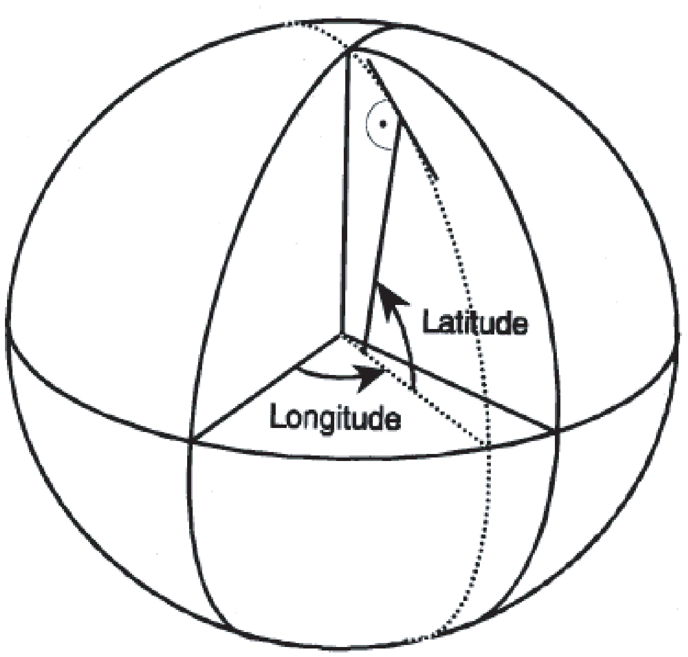


Figure 5.1: Description of a point as two coordinates

5.2 Universal Transverse Mercator (UTM)

Universal Transverse Mercator (UTM) coordinates define two dimensional, horizontal, positions. UTM zone numbers, see the horizontal lines in Figure 5.2, designate 6 degree longitudinal strips extending from 80 degrees South latitude to 84 degrees North latitude. In addition, UTM zone characters designate, see the vertical lines in Figure 5.2, 8-degree zones extending north and south from the equator. There are special UTM zones between 0 degrees and 36 degrees longitude above 72 degrees latitude and a special zone 32 between 56 degrees and 64 degrees north latitude.

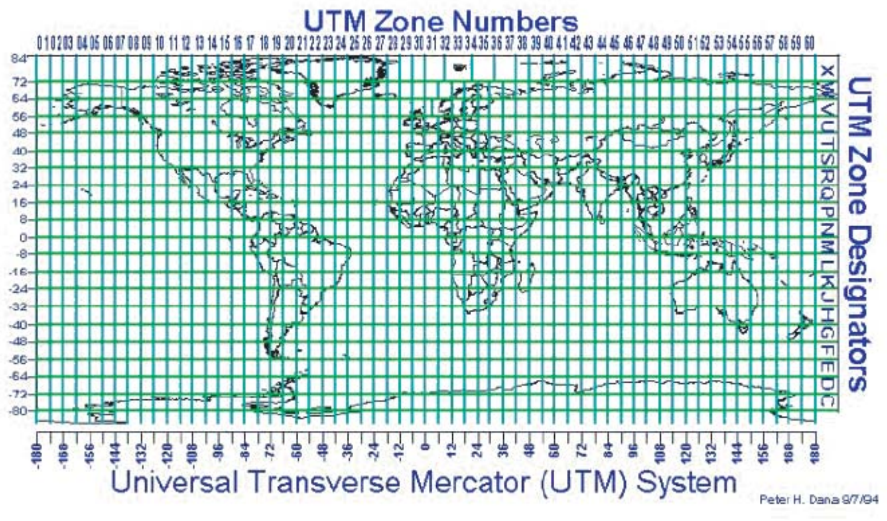

UTM Zone Numbers
UTM Zone Designators
X W V U T S R Q P N M L K J H G F E D C
Universal Transverse Mercator (UTM) System
Peter H. Dana 9/7/94

Figure 5.2: World map divided into UTM zones

5.3 Rikets Trangulering 90 (TR90)

The TR90 coordinate system is a specialized version of the Transverse Mercator coordinate system. It is made specifically for Sweden and has some special properties which makes it useable only in Sweden. More about this system can be found at [Geo04b] and [Eri01].

5.4 Summery - Three coordinate systems

When working with location-aware applications, the choice of coordinate system the system is going to use is an important part. To ease the transformation from real world coordinates to computer screen device coordinates the prototype thats being developed in these project will use UTM or RT90. However, our framework should be able to run with every available coordinate system.

Chapter 6

Selection of technologies

In the project performed during fall 2003 [GS03], several technologies for developing software were presented. After the technologies were presented, an evaluation was conducted to choose the most appropriate technologies.

This project is a continuance of the fall project, and a new evaluation of technologies has not been conducted. This project will use the same technologies as the fall project, presented in Table 6.1.

Modelling language	UML
Database	MySQL
Web Server	Tomcat as servlet container
Programming language	Java

Table 6.1: Selection of technologies

This master thesis demands development of systems that have well-designed graphical user interfaces. For this purpose, Visual Basic (VB) and the development tool Visual Studio 6.0 will be used. By using Visual Studio it is easy to develop graphical user interfaces by using the drag and drop functionality. In addition, the resulting program obtains a typical Microsoft look and feel that many users are familiar with.

Part III

Construction and Implementation

This part contains documentation of the framework developed. It starts by presenting an overall explanation of the system. Then it continues by presenting the requirement specification, the system architecture, implementation details and testing documentation.

Chapter 7

Overall explanation of the system

This chapter gives a brief introduction to the framework. The goal is to quickly present the framework to give the readers of this report an overview of the system and make the rest of this report easier to read.

7.1 The framework

The framework for developing location-aware tour guides consists of three parts, of which only two are implemented:

- A Runtime System
- A Creator Tool
- A Statistical Tool (not implemented)

The Runtime System is located on a server. This system is used during the guided tours to deliver information to the users' handheld devices. All the information about the attraction is stored in a database on this server. The server also holds the location of the users located in the attraction area. The information on the server is transferred to the handheld devices while the user is moving around, adjusted to the users location on the attraction area.

The framework supports the use of various types of handheld devices, and has been demonstrated both with cellphone and PDA. To locate the users, the Cordis RadioEye is used. Figure 7.1 demonstrates a simplified model of the architecture of the system.

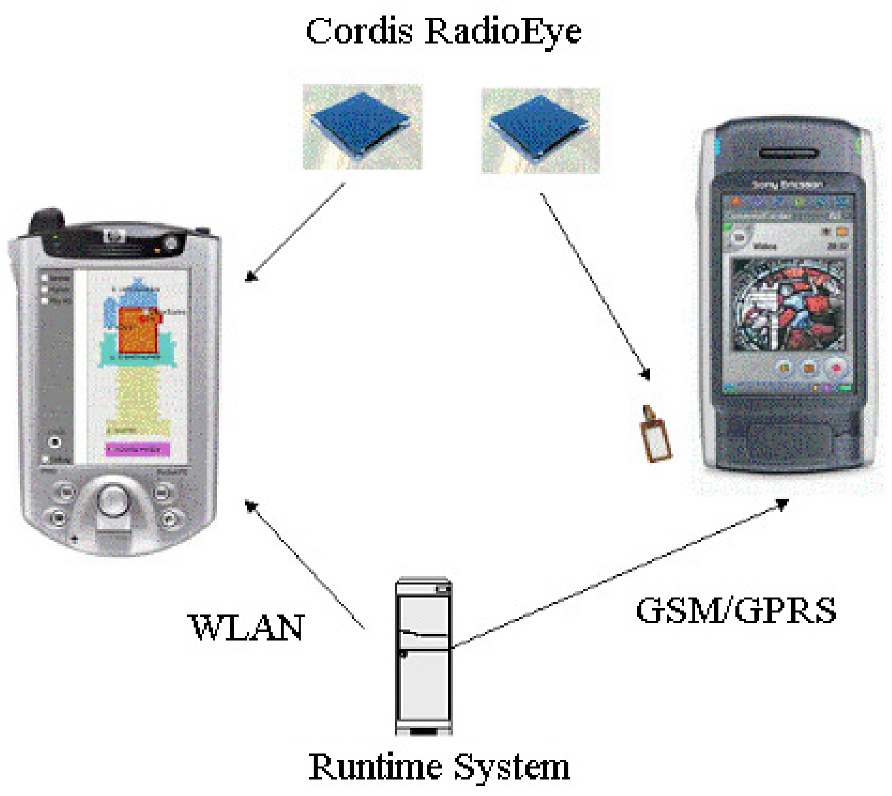


Figure 7.1: Simplified model of the architecture

The Creator Tool is a system for registering information about a new tour guide that is going to be developed, or editing information about an already existing tour guide. For instance, in some cases it is desirable to be able to move the location of a sight, like a picture or a sculpture, from one place to another. By using the Creator Tool, it is easy to both develop and configure the tour guides.

The users movements and actions are stored in a log, and the Statistical Tool shall be used to generate various statistics about the users interests in the attraction based on this information. Each user receives a unique user id, hence now personal information is logged.

Chapter 8

System Requirements

The system requirements are developed based on input from Klipp and Lim Media and Telenor R&D. In addition, the requirements have been influenced by information gathered during the prestudy.

This chapter intends to be a structural description of the requirements to the framework, and it will be used during design and implementation. The chapter is divided into three main sections; an overall explanation of the system, the functional requirements and the non-functional requirements.

8.1 Overall explanation of the system

The development of location-aware applications is a growing discipline, and new technologies for positioning determination are constantly being developed. As new applications arise, the need of a framework for developing these applications seems more and more necessary.

Such a framework should be a client-server solution, where mobile devices are the clients and necessary data is stored on the server.

The ideal framework would be a system where it is possible to both make the graphical user interface on the client side and provide the system with the necessary location related information on the server side. Since there already exists advanced tools for making good graphical user interfaces on the client side, i.e Flash MX, this master thesis is not going to deal with that aspect. Instead, this thesis focuses on the server side.

The generic framework shall consist of three main parts; a *Creator Tool*, a *Statistical Tool* and a *Runtime System*. The Creator Tool's task is to configure and create new tour guides by editing information stored in a database. The user of this tool has to be a competent person with good computer skills. The Statistical Tool is for the staff of the attraction. This tool shall be used to generate various statistics about the users interests in the attraction based on information stored in a log. The Runtime System is the system that is providing the mobile devices with location based information during a guided tour.

The framework's goal is to make it easy to develop new location aware-applications, reduce the time to market since the applications don't have to be developed from scratch, and reduce the costs since the development time is reduced.

8.2 Functional requirements

The functional requirements describe the functionality or services that the system is expected to provide [Som01].

There are two kinds of stakeholders that will be using the framework; the developer of the location-aware applications and the staff of the attraction. The stakeholders have different interests in the system, and the requirements are therefore organized according to the stakeholders.

8.2.1 The developer of the location-aware applications

F-1 and F-2 are requirements for the Creator Tool.

F-1 Register new tour guide
The framework must have a graphical user interface with a connection to the database. By using this interface, the developer should be able to transfer the necessary input to the database to generate a new guide. The input consists of location information about the attraction area.

F-2 Make changes to existing tour guide
The developer has to be able to alter an existing guide and make changes if required. These changes can be a new location, or maybe removal, of an object, or substitution of information about one exhibition with another.

F-3 to F-10 are requirements to the Runtime System.

F-3 Simulate runtime system
There should be a simulation service available. This service returns simulated location data to the client, and will by this ease the development of client applications.

F-4 Support map functionality
If the clients have a map available, the framework shall provide the clients their positions in correct map coordinates. If the map consists of several zones, the system shall give information about the area the user are located in.

F-5 Support hotspot[1] information
The framework shall be able to make the clients aware that a hotpost is nearby.

[1] A *hotspot* is an attraction, ie ”Korsalteret” in Nidarosdomen

F-6 Support a service to locate friends
There shall be a service available to locate friends on the area. To use this service, all the parties must agree that the other users in the group can state their location.

F-7 Support a service to send messages
There shall be possible to send messages between the handheld devices to friends located on the area. To use this service, all the parties must agree that the other users in the group can send messages to them.

F-8 Support dynamic menus
The user shall be able to choose what kind of information he is interested in receiving. That means that the user shall be able to state his preferences, and the menus on the user's display shall be adjusted to these.

F-9 Support predefined tracks
Some predefined tracks shall be available to the user. These tracks shall help the user to navigate round the attraction area.

F-10 Support a service to maintain a log
The user's actions and movements shall be stored in a log, but no personal information shall be stored.

8.2.2 The Staff

The staff represents the persons with special interest in the attraction, e.g people who work there or people with financial interests in the attraction.

F-11 is a requirement for the Statistical Tool.

F-11 Support a service to show statistics
By using the information in the log, it shall be possible to generate different statistics, e.g which attraction is most frequently visited and which attraction is least frequently visited.

8.3 Non-functional requirements

The non-functional requirements, as the name suggests, are those requirements that are not directly concerned with the specific functions delivered by the system [Som01]. Many non-functional requirements relate to the system as a whole rather than to individual features, and not all non-functional requirements are concerned with the software of the developed system.

The non-functional requirements have an important role in the framework. Requirements like usability, maintainability and extendibility are vital to the system's successfulness among location-aware system developers.

The non-functional requirements are divided into three categories; product requirements, organizational requirements and external requirements.

8.3.1 Product requirements

The product requirements are requirements that specify product behaviour [Som01]. This section classifies the product requirements under its belonging quality attribute. The quality attributes are defined based on Tom Gilb's definitions in [Gil04].

Each product requirement is given a priority level to locate the requirements that need most attention, and also to point out witch requirement that is prioritized if conflicts between the requirements are revealed. The priority levels are *high*, *medium* and *low*.

Usability - How easy a system is to use

NF-1 Experienced computer users shall be able to set up and configure the Creator Tool to run as a specific location-aware application without more support than is given from the systems own documentation.
Priority Level: *High*

NF-2 The Runtime System shall return understandable information both when the client application states correct and wrong input.
Priority Level: *High*

Portability - The cost to move the system from location to location

NF-3 The framework shall run on Windows 2000, but it shall be possible to extend the system to run on newer versions of Windows if desired.
Priority Level: *Medium*

Extendibility - The cost to extend the framework

NF-4 The Runtime System shall have an infrastructure that makes it possible to add new positioning systems, e.g WLAN, IrDa, GPS or GSM positioning.
Priority Level: *High*

NF-5 There shall be possible to use different transmission protocols in the runtime environment, e.g. WLAN, GSM, IrDA and Bluetooth.
Priority Level: *Medium*

Performance - The responsiveness of the system

NF-6 The Runtime System should not use more than 10 seconds before it returns a response to a request from a mobile client.
Priority Level: *Low*

Comment: The responsiveness will vary from positioning system to positioning system. It is out of scope of this master thesis to try to improve the responsiveness of the various systems.

Maintainability - Resources required to repair an unreliable system

NF-7 The framework shall be module-based.
Priority Level: *High*

Resource saving - Financial and time saving

NF-8 The total time used to develop a location-aware guide by using the Creator Tool shall be less than when the system is developed from scratch.
Priority Level: *High*

Installability - The cost to install the system

NF-9 For experienced user it should take less then 2 days to install and configure the Runtime System so that it can give location information about devices using preinstalled sensor.
Priority Level: *Low*

Security - The confidence that the system has suffered no harm

NF-10 The staffs and the users shall not be able to edit the content stored in the database; only authorized personnel shall have access to critical parts of the framework.

8.3.2 Organizational requirements

Organizational requirements are requirements that relates to polices and procedures used during the development of the framework (standards, programming languages and design methods) [Som01].

NF-11 Programming language
Server side programming for the Runtime System shall be done in Java. The Creator Tool and the Statistical Tool can be programmed in desirable languages.

NF-12 Modelling language
The system architecture and design shall conform to the Unified Modelling Language, UML. The Entity-Relationship (ER) diagram shall be used for database modelling.

NF-13 Database system
All data shall be stored in a MySQL database.

NF-14 Standards used
Transmission of data between main components in the Runtime System shall follow the XML standard. That means that all handheld devices that have the ability to interpret XML data can be used on the client side.

8.3.3 External requirements

These requirements will cover all requirements that are derived from external factors to the system and its development process. This includes legislative requirements that must be followed to ensure that the system operates within the law and also ethical requirements [Som01].

NF-15 Legislative requirements
The system shall not disclose any personal information about users apart from their name and reference number to the operator of the system (this requirement concerns the logging issue).

Chapter 9

Architecture

There exist several different definitions of software architecture, all of which are trying to capture the same content. In [LB03], software architecture is described as: *Structure(s) of the system, which comprise software components, the externally visible properties of those components and the relationships between them*.

Two articles about software architecture are the foundation for the architecture chapter in this master thesis. These articles are ”The 4+1 view Model of Software Architecture” [Kru95] and ”IEEE Recommended Practice for Architectural Description of Software-Intensive Systems” [Boa00]. These articles describe what should be documented in an architecture and how it should be done.

The architecture chapter is divided into two main parts. Section 9.1 gives a brief introduction to software architecture documentation the way it is presented in [Kru95] and [Boa00]. Section 9.2 gives a detailed description of the architecture of the generic framework.

9.1 Introduction to architecture documentation

The primary focus in software architecture is the identification of important properties and relationships. The IEEE 1471 standard recommends that an architecture document consist of five parts. The first part is identification of stakeholders and concerns; the second part is selection of architectural viewpoints; the third part is description of the architectural views; the fourth part discusses the consistency among the architectural views and the fifth part gives a rationale to the architecture.

A *stakeholder* is a person with interest in, or concerns relative to, the system. Stakeholders include, among others, acquirers, users, architects, developers and evaluators. A system has one or more stakeholders. *Concerns* are those interests that pertain to the systems development, its operation or any other aspects that are critical or otherwise important to one or more stakeholder. Concerns include system considerations such as performance, reliability, security, distribution and evolvability.

The terms *view* and *viewpoint* are central to the IEEE standard. A viewpoint determines the languages to be used to describe the view, and any associated modelling methods or analysis techniques to be applied to these representations of the view. A view is the actual representation of a particular system from a single perspective. One viewpoint can consist of one or several views.

In [Kru95], five viewpoints are presented. These viewpoints are *logical*, *process*, *physical*, *development* and *scenarios*. The *logical viewpoint* describes the object model of the design and gives the different classes of the system (when an object-oriented design method is used), the *process viewpoint* captures the concurrency and synchronization of the design, the *physical viewpoint* describes the mapping of the software onto the hardware and reflects its distributed aspect, while the *development viewpoint* describes the static organization of the software in its development environment. The elements in these four views can be shown to work together seamlessly by the use of a small set of important *scenarios*.

9.2 The architecture of the generic framework

The architecture of the generic framework presented in this chapter is based on the IEEE standard [Boa00], described in section 9.1.

9.2.1 Identification of stakeholders

The stakeholders in the system are users, framework developers, end system developers, testers, staff and maintainers:

- *The user* represents the end user who is going to run the location-aware application on a handheld device.
- *The framework developer* represents the people implementing the generic framework.
- *The end system developer* represents the people who is going to configure the framework to function well with a specific location-aware client application.
- *The tester* represents the people who are going to perform the system test in the finishing phase.
- *The staff* is persons with special interests in the tourist attraction where a specific location-aware system is used; e.g people who work there or people with financial interests in the attraction.
- *The maintainer* is the person who is responsible for maintaining the system after its installation date.

9.2.2 Selection of architectural viewpoints

The architecture of the framework is described by using the five viewpoints presented in section 9.1.

Scenario viewpoint

The scenario view describes different scenarios relative to the framework. It also shows which components that deals with the various scenarios.

Stakeholders:	User, end system developers and staff.
Concerns:	Which part of the system serves different user scenarios?
Notation:	UML Use Case diagram.

Physical viewpoint

The physical view describes how the various hardware components are organized in the framework.

Stakeholders:	Framework developers, end system developers, staff and maintainers.
Concerns:	How are the physical communications interconnected among system components? How is the hardware in the overall system organized?
Notation:	UML deployment diagram.

Development viewpoint

The development view focuses on the actual software module organization on the software development environment.

Stakeholders:	Framework developers, end system developers, maintainers.
Concerns:	How is the organization of the actual software modules?
Notation:	UML component diagram.

Logical viewpoint

The logical view describes how the system is decomposed into elements and the relations between those elements.

Stakeholders:	Framework developers, end system developers, maintainers.
Concerns:	What are the main classes in the system and how are they related?
Notation:	UML class diagram.

Process viewpoint

The process view describes how information flows between the components of the system.

Stakeholders:	Developers, maintainers, testers.
Concerns:	What are the running processes of the system?
	How do the processes communicate?
Notation:	UML sequence diagram.

9.2.3 Architectural views

As described in the system requirements, the framework is divided into three subsystems; the Statistical Tool, the Creator Tool and the Runtime System. The staff on the attraction will use the Statistical Tool and the persons configuring the specific location-aware system will use the Creator Tool. The Runtime System is used to give the clients information adjusted to their locations.

Scenario view

The scenario view is divided into three sections. Each section deals with one of the framework's subsystems.

Statistical Tool
Figure 9.1 shows the staff using the Statistical Tool. Staffs are interested in seeing statistics about the users actions in the attraction area. Their scenario is:

1. View statistics

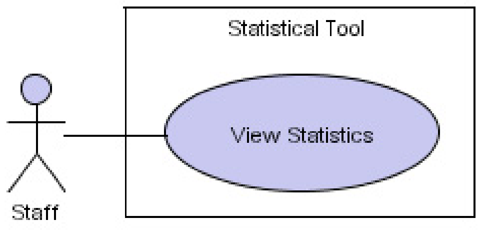


Figure 9.1: Scenario view - Statistical Tool

Creator Tool

Figure 9.2 shows the scenarios for the end system developer during development of tourist guides. The scenarios are:

1. Create new tourist guides
2. Edit existing tourist guides

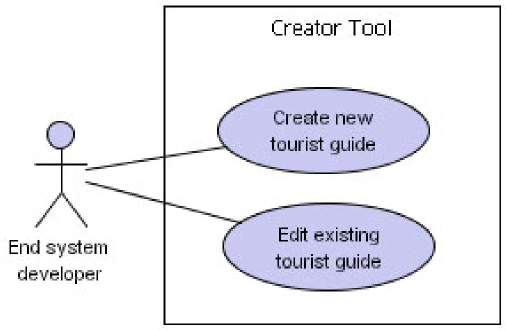


Figure 9.2: Scenario view - Creator Tool

Runtime System

The end user is interested in different type of information from the Runtime System. Figure 9.3 shows five different scenarios for the end user. These scenarios are:

1. Get position information
2. Locate friends
3. Send/get messages
4. Get dynamic menu information
5. Get predefined tracks

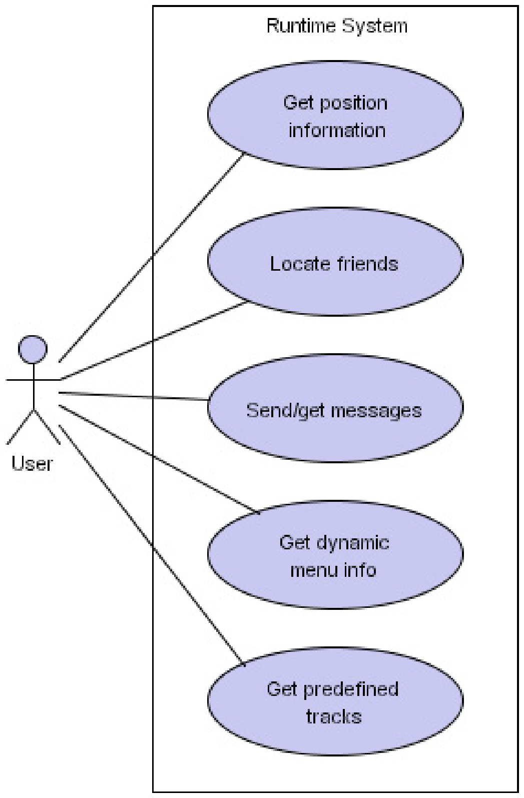


Figure 9.3: Scenario view - Runtime System

Physical view

This section is also divided into three subsections based on the three subsystems of the framework.

Statistical Tool

The Statistical Tool, Figure 9.4, connects to the database via a local area network. The tool will give the staff various statistics about the users interests in the attraction based on the information stored in the log.

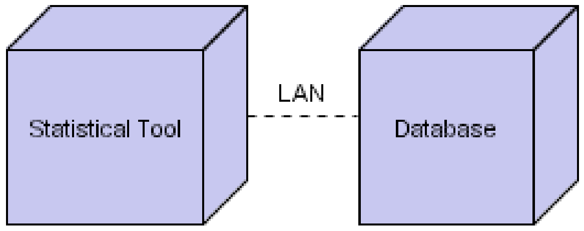


Figure 9.4: Physical view - Statistical Tool

Creator Tool

The Creator Tool, Figure 9.5, will as the Statistical Tool connect to the database by means of a local area network. By using the Creator Tool, the end system developer can create new or edit existing location-aware applications.

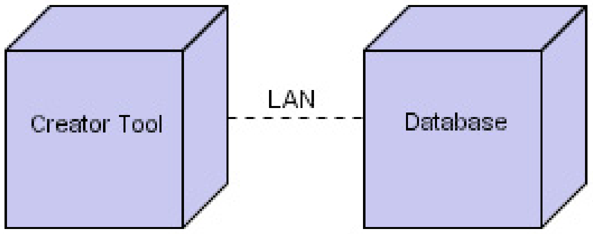


Figure 9.5: Physical view - Creator Tool

Runtime System

The Runtime System includes physical clients like PDAs, cellphones and Tablet PCs, as shown in Figure 9.6. These clients run the location-aware applications, and they use the location information delivered by the Runtime Server to give the user adjusted information based on their location. The database is storing the necessary location information.

The client's position is given from an external Location Server (LS). The LS connects to several positioning systems; examples are Cordis RadioEye, GPS and IrDa.

The system also has a File Server where the media files are stored. The files have to be stored on a File Server because the mobile clients often have small storage room, and the files will be transferred to the clients on request. The media files are the files containing the presentations of the various objects in the attraction area.

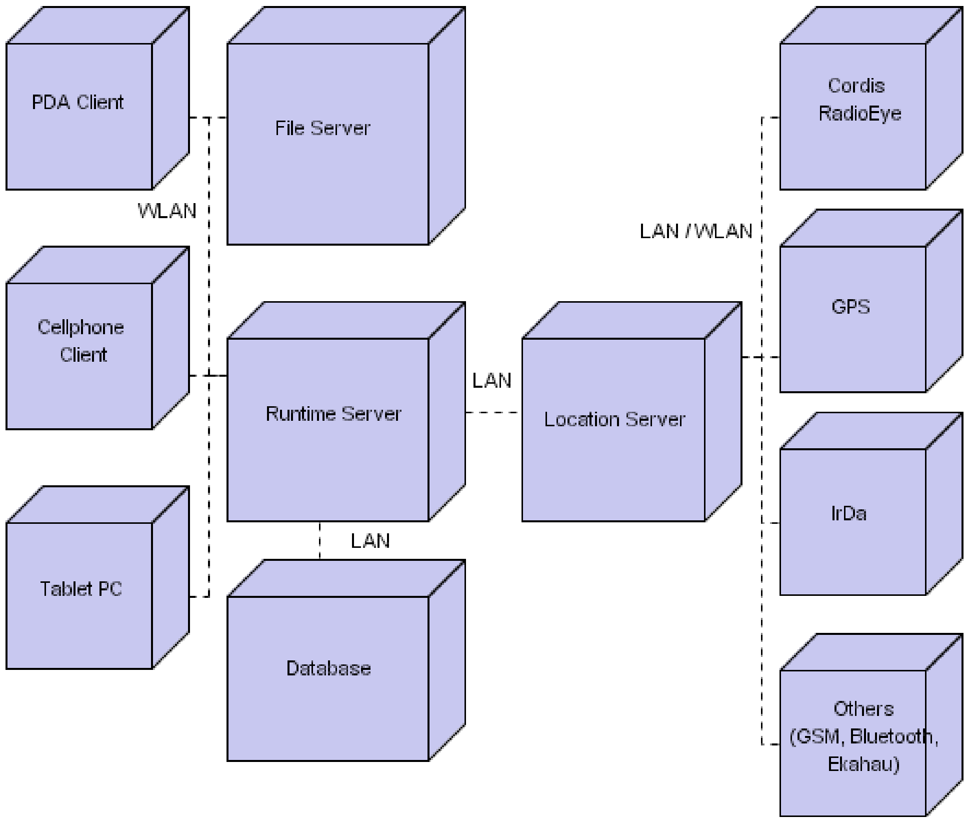


Figure 9.6: Physical view - Runtime System

Development view

The framework consists of three main software components; the Statistical Component, the Creator Component and the Runtime Server Component. In addition to these components, there is a Location Server that states the mobile devices locations, and a database where all static system information is stored.

In Figure 9.7, a 3-layered development view is showing the components in the generic framework. The database is connecting all the components together.

To follow a standard and make it easy to swap old components with new, XML is used in the communication between the components. SQL is used to communicate with the database.

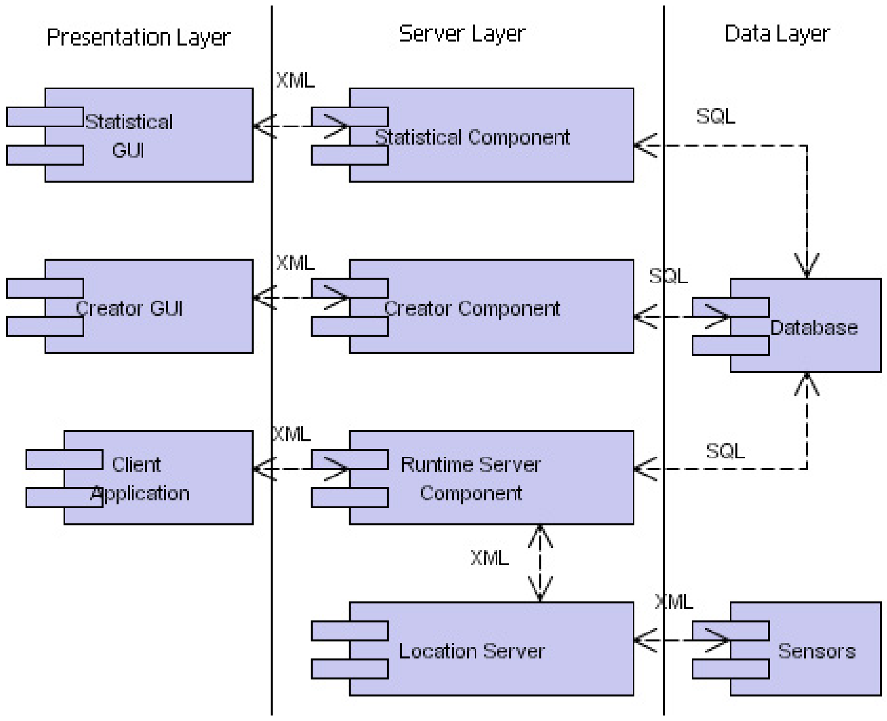


Figure 9.7: Development view - Overall framework

Logical view

The three subsystems are presented in separate logical views.

Statistical Tool

The Statistical Tool is divided into four main classes, shown in Figure 9.8. The GUI class presents the different services available to the staff; the LogTool class is responsible for calculation and manipulation of the data stored in the database; and the DbManager and DbLog handles the database issues.

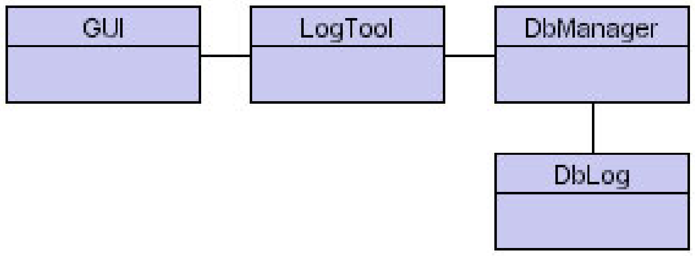


Figure 9.8: Logical view - Statistical Tool

Creator Tool

The Creator Tool contains four main classes, shown in Figure 9.9. The GUI class presents interfaces for creating and editing the location-aware systems. The LocSysEditor class is responsible for validation of the input data and communicating with the DbManager. The database classes are responsible for updating the database with the information the user has given.

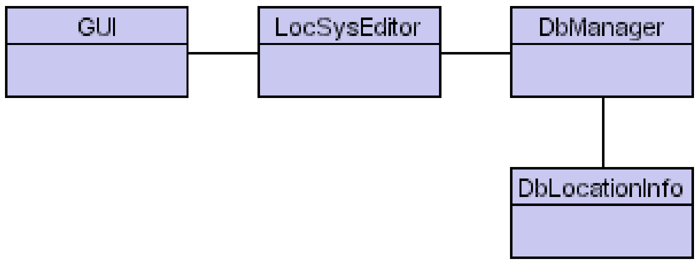


Figure 9.9: Logical view - Creator Tool

Runtime System

The whole Runtime System consists of several different components as described in the physical view section. The logical view described in this section correspond to the Runtime Server in Figure 9.6

The Runtime Server consists of 20 main classes, shown in Figure 9.10. The system is communicating with the client applications through the *GLocServlet* class. The communication consists of XML data, and the *XMLTransformer* interprets and transforms the information that is sent between client and server.

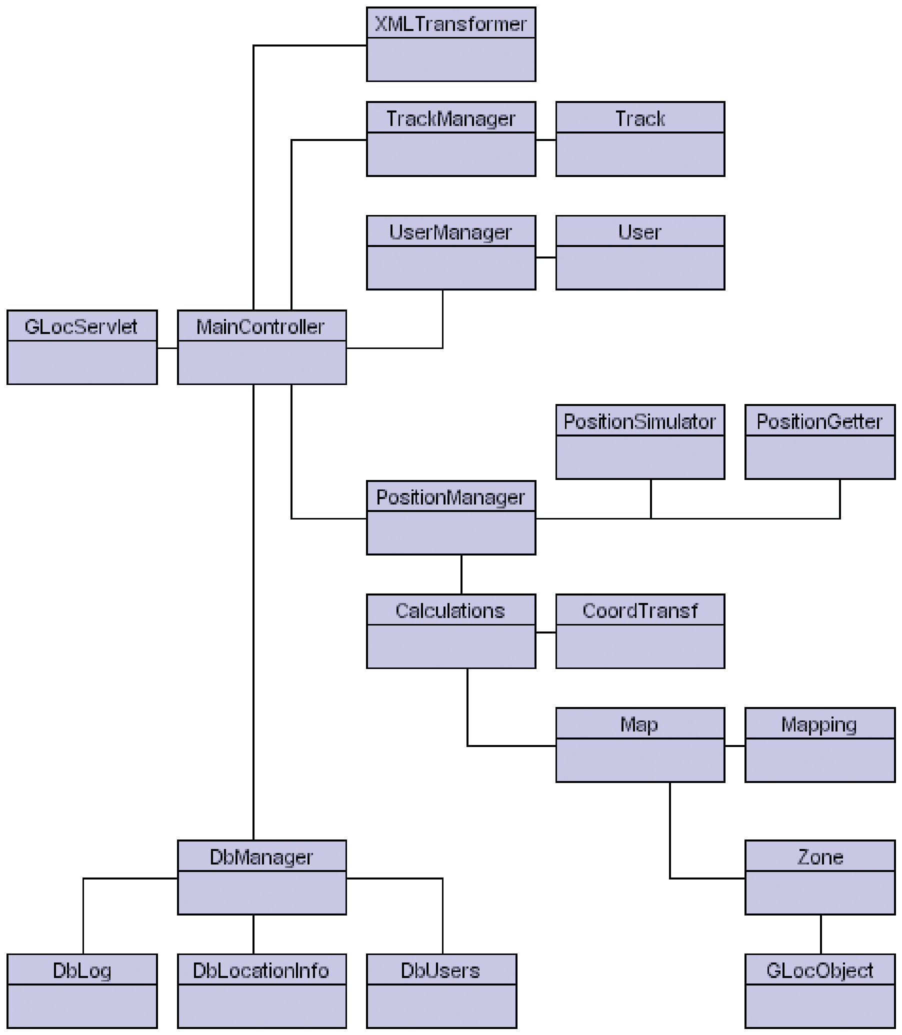


Figure 9.10: Logical view - Runtime Server

In the Runtime Server's architecture, a centralized control model is used. All information flows through the *MainController* class. This centralized control makes it easy to analyse control flows and find out how the system will respond to particular inputs. It also makes it easy to substitute the servlet class with another class for handling the communication with the clients.

The *UserManager* class is responsible for maintaining information about the users. This task includes storing the users last position and deciding wether a person is allowed to communicate with another person.

The *PositionManager* class is responsible for returning the user's position, adjusted to the type of mobile device he is using. *Calculations* and *CoordTransf* are classes that contribute to transfer universal coordinates to map coordinates. The *Zone*, *Object*, *Map* and *Mapping* classes are used to store information about the attraction.

The *TrackManager* class is responsible for keeping information about the predefined tracks that are available. Each track has a unique name, so the client application can either request all tracks or just one particular track.

The system can be used in two different ways, at runtime or at development time. At runtime, the system will use the *PositionGetter* class and return real position information fetced from the LS back to the client. At development time, the system will use the *PositionSimulator* class and return simulated position information to the client. That means that during development of the client applications, the developer can receive test data to simulate how the system will work.

The *DbManager* class is responsible for all the communication with the database. This class uses the *DbLog*, *DbLocationinfo* and *DbUsers* classes to perform tasks when requested.

Process view

The three subsystems are presented in separate process views.

Statistical Tool

Figure 9.11 demonstrates the sequence diagram for the use of the Statistical Tool. During startup, the *init()* method is called. This method takes care of the presentation of the graphical user interface and establishes a connection to the database. After the user has made his selections of what kind of statistics he is interested in, the *getStatistics()* method is called. This method takes care of the collecting of information from the database, and the information is then analysed. Finally, the statistics are presented to the user in a proper manner.

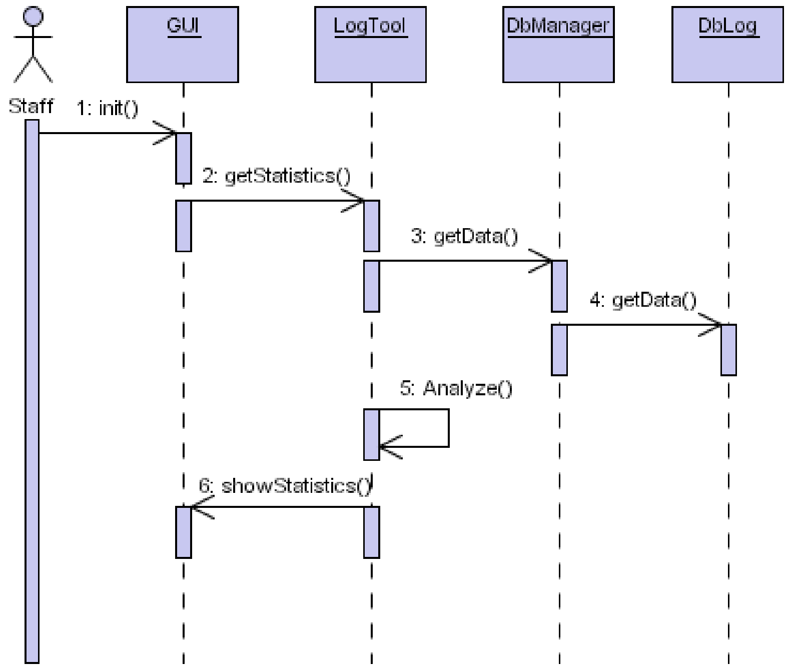


Figure 9.11: Process view - View statistics

Creator Tool

The Creator Tool can be used in two ways, to create a new location-aware application or edit an existing location-aware application.

Figure 9.12 demonstrates the sequence diagram for creating a new location-aware application. The *init()* method is responsible for presenting the graphical user interface and to establish a connection to the database. After the developer has given the desired information, the *createApp()* method is called. This method calls another method to validate the input data from the user, and makes sure that the input data is written to the database by calling the *write()* method. The *write()* method sees to that the methods *createSQL()* and *performSQL()* are called.

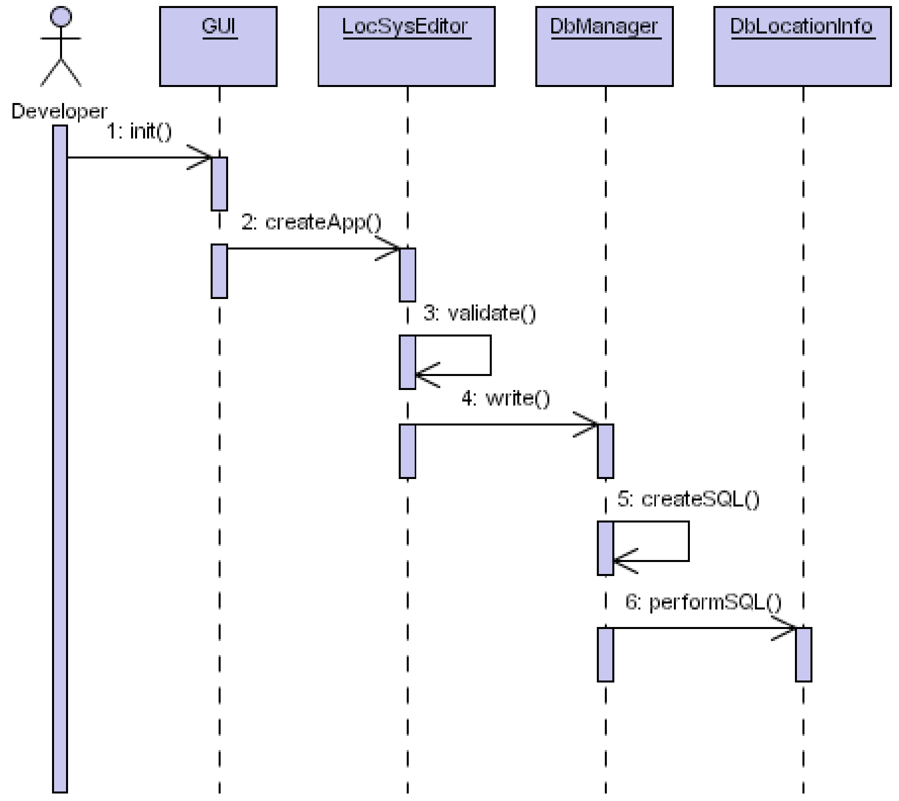


Figure 9.12: Process view - Create new location-aware application

Figure 9.13 demonstrates the sequence diagram for editing an existing location-aware application. This sequence diagram is similar to Figure 9.12. The only difference is that the *createApp()* method is substituted with the *editApp()* method.

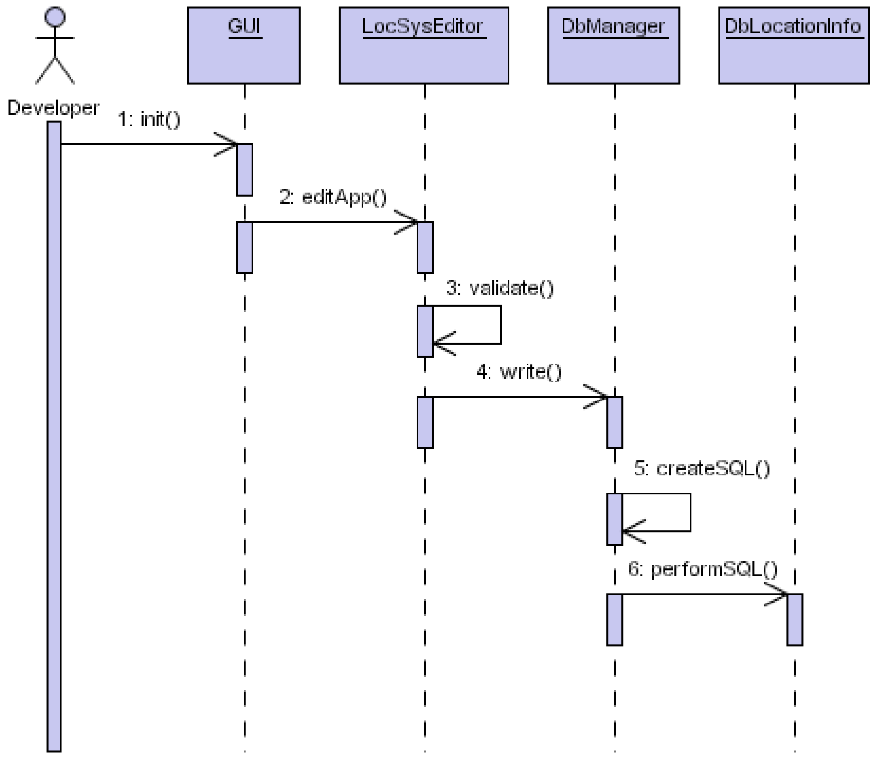


Figure 9.13: Process view - Edit existing location-aware applications

Runtime System

During runtime, different events can take place. The clients can request for location information, exchange messages with friends or find out where friends are located. Clients can also request for dynamic menu information or they can request for information about predefined tracks. The sequence diagrams presented in this section correspond to the class diagram of the Runtime Server described in Figure 9.10.

Get location information
Figure 9.14 shows the sequence diagram for sending a request to find users locations and calculate nearby objects. To contact the system, the client application calls the *doGet()* method on the servlet. The request is then forwarded to the *MainController*, which is responsible of finding the correct object to accomplish the task. In this case, the *PositionManager* is responsible of the accomplishment of the get location information task. Before the position information is returned to the user, it is transformed to XML using the *XMLTransformation* object.

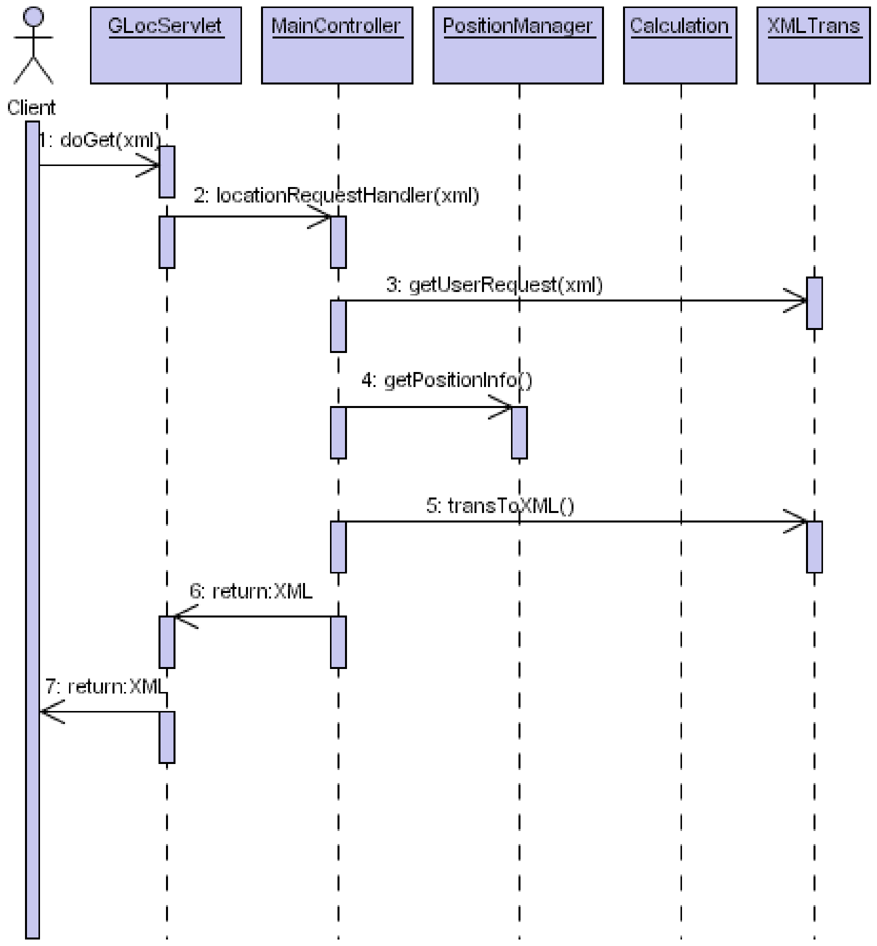


Figure 9.14: Process view - Get location information

Locate friends
The Runtime Server contains a service to locate friends on the attraction area. The client contacts the *GLocServlet* by its *doGet()* method. The *GLocServlet* contacts the *MainController*, which will forward the task to the *UserManager*. The *UserManager* examines wether the user is allowed to locate the person. If permission is granted, the friends positions are returned to the user. This sequence diagram is shown in Figure 9.15.

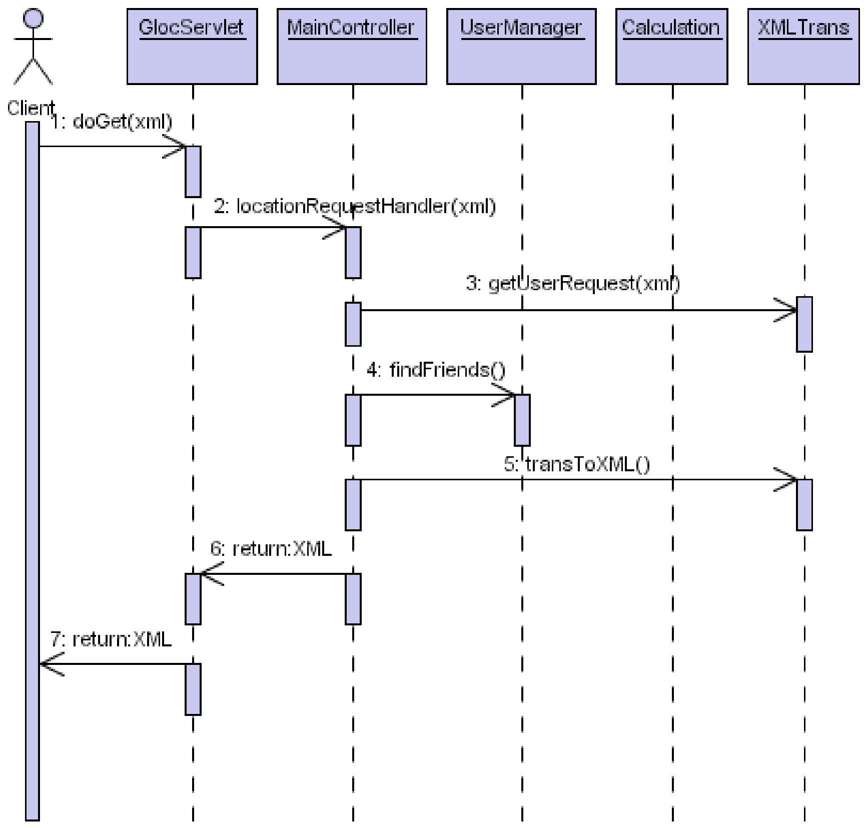


Figure 9.15: Process view - Locate friends

Exchange messages
When exchanging messages, there are two possible events. The user can either send, Figure 9.16, or receive Figure 9.17 a message. The mobile clients are often not able to listen to incoming events. Therefore, the messages have to be received through a request. Each sent message is saved on the server in a user object, and users recieve the messages on demand.

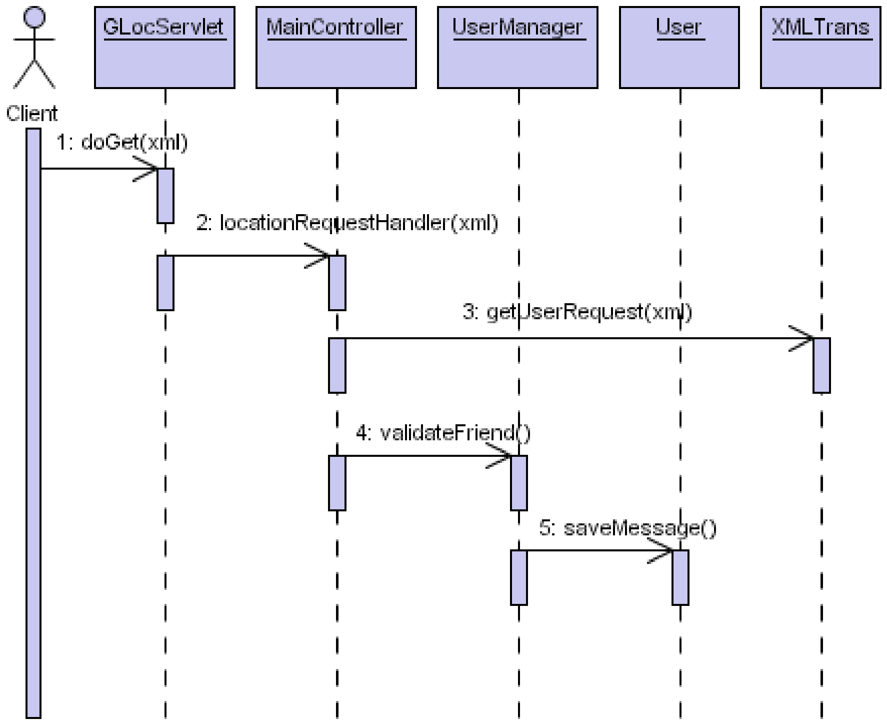


Figure 9.16: Process view - Send message

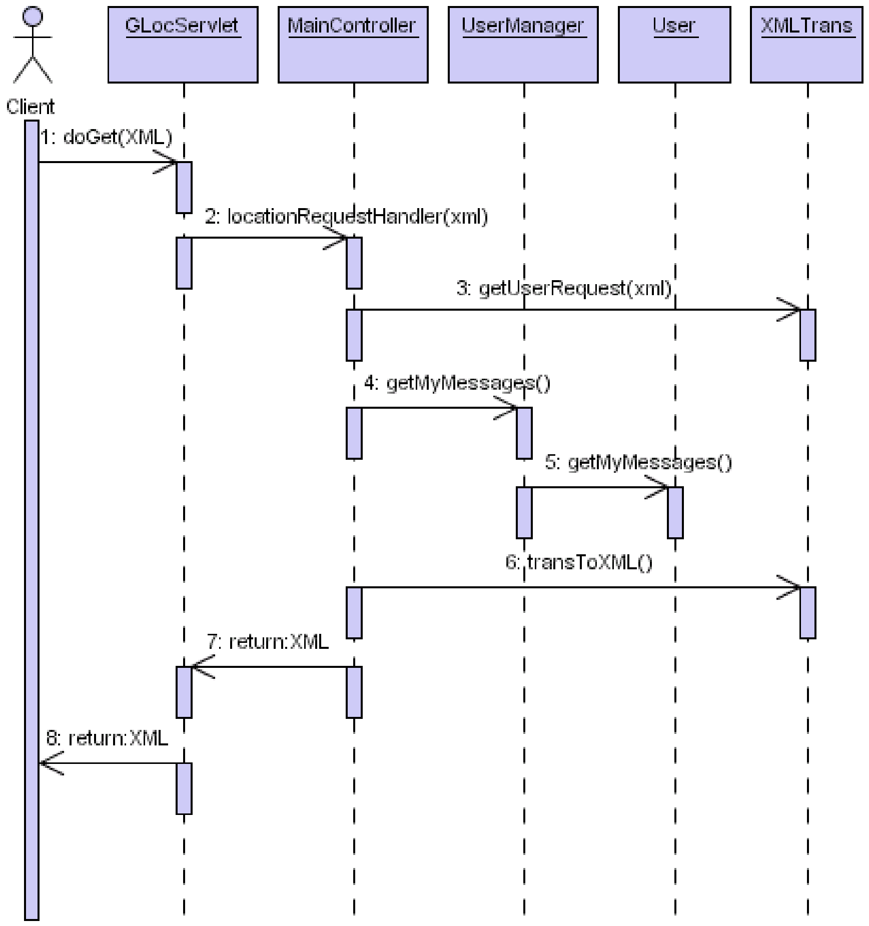


Figure 9.17: Process view - Get messages

Get dynamic menus
Figure 9.18 describes the sequence of events taking place when client applications call for dynamic menus. First, the Runtime Server fetches the user's preferences by using the *UserManager*. Afterwards, the system iterates through the objects by using the *PositionManager*, and returns every object that have a match with the user's preferences.

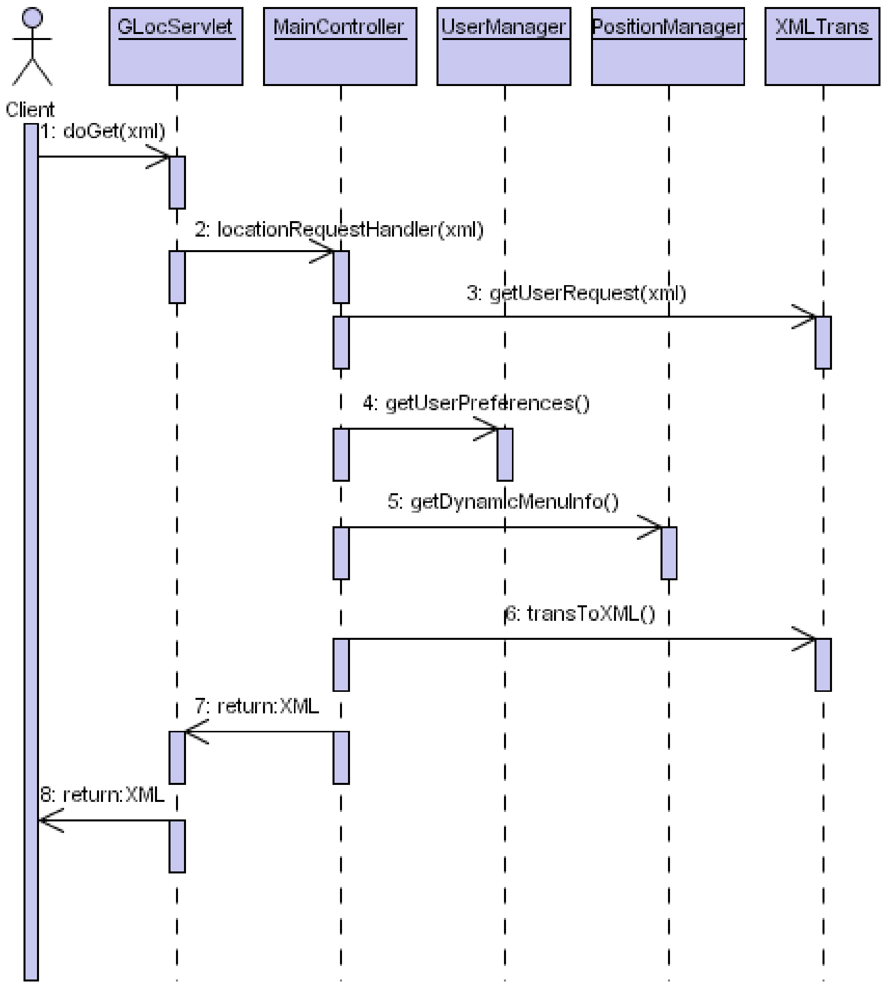


Figure 9.18: Process view - Get dynamic menus

Get predefined tracks
Figure 9.19 describes the sequence of events taking place when client applications call for the get predefined track service. After the Runtime Server has interpreted the request from the client, the *getTracks()* method in class *TrackManager* is called. This method can either get all registered tracks, or it can get one particular track specified by the user.

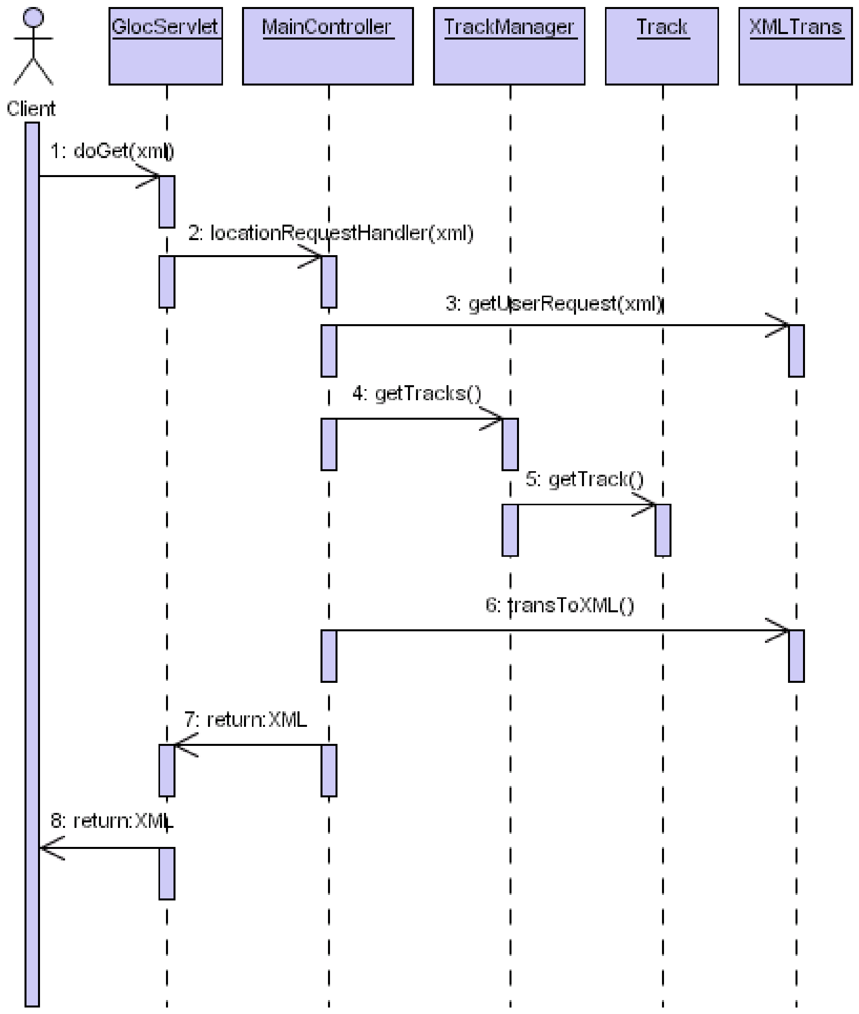


Figure 9.19: Process view - Get predefined tracks

9.2.4 Consistency among architectural views

The architectural description consists of five different views. The views are capturing the same system, but from different perspectives.

The physical view describes how the physical hardware components in the framework are organized. The development view is a more detailed description than the physical view, where the focus is on the software module organization inside the different hardware components. The logical view is an even more detailed description than the development view, where some modules are decomposed into classes. Finally, the process view shows the interaction between the classes presented in the logical view.

Even though the views get more and more detailed, it doesn't mean that for example the physical view is unnecessary. The views present information about the framework in different perspectives, and the information in the different views are interesting to different stakeholders. All the views united constitute an adequate architectural description.

By examining the views of the different subsystems presented in this chapter, you will find this relation between all the views. In addition, each scenario corresponds to one process view, and the scenario view is used to bind the other views together to make the architecture complete. Therefore, no inconsistency in the architecture is found.

9.2.5 Architectural rationale

The architectural rationale discusses the advantage and disadvantages with the architectural choices.

Security

The generic framework is divided into three main components, all of which are running as independent subsystems. The reason is that different stakeholders only have interests in parts of the system. This dividing keeps the security level high, because a stakeholder using one subsystem is unable to enter a part of the system he is not authorized to use. For example, when the staffs are using the Statistical Tool, they cannot get in contact with the same information as the developers using the Creator Tool. This keeps the staffs unable to edit or delete any of the information stored in the database, they are just allowed to read from the log database.

Performance

Figure 9.7 shows that a three-tier architecture is used. The client layer is concerned with the presentation of the information, the server layer is concerned with logic and calculations and the data layer is concerned with the database operations.

The advantage by doing the calculations on the server is that the client applications will not suffer from any overhead. The handheld devices have limited processing capacity, and the applications will run smoother when its only concerns are the presentation.

In the Runtime Server, a centralised control model architecture is used. All information flows through the *MainController* class. This architecture makes it simple to analyse control flows and to work out how the system will respond to particular inputs. It also makes it easy to substitute the servlet class that is responsible for the communication with the clients. A drawback with the centralised control model is that the central component can be a bottleneck. If the *MainController* fails, the whole system will fail. Also, if there are any errors in the *MainController* or the *MainController* runs slowly, there will be performance and reliability problems in the system. Therefore, when implementing this class, great caution must be taken.

Maintainability

The system uses object-oriented design methods to make it easy to maintain. When working object-oriented instead of function-oriented, components are more loosely connected and easier to reuse.

In addition, the framework is divided into independent components. These components use XML to communicate which makes them easy to exchange with new components. A drawback with the use of XML data may be that the throughput can be reduced since the parsing of XML data is more time-consuming than the interpretation of plain text.

Availability

The wireless network is a critical factor to the availability of the system. The network has to cover all areas that are relevant to the system, and it is critical that the clients don't loose the connetction to the network during a guided tour. All the media files are stored on an external file server and will be transferred to the client on request. That means that a client without connection to the network will not be able to recieve any information about the various objects. Therefore, the client applications should contain some of the media files of the main attractions on the handheld devices in case of no connection with the network.

In case of a breakdown of the Runtime Server, no data will be lost. All the information about the attraction is always stored in a database, and in addition all the information about the users are also stored in this database. Before the Runtime Server is ready to recieve new requests after a breakdown, all this information have to be read from the database. After the user has ended his session, the information about the user in the database can be deleted.

Extendibility

The architecture of the system makes it easy to both extend new type of sensors and new clients. The LS handles all location issues, so the Runtime Server doesn't have to be concerned with this issue. Since the communication between server and client uses

XML data, the only requirement to the client is that it is able to interpret XML data. In addition, XML can be transferred over different protocols.

Chapter 10

Database modelling

The framework uses a database with ten tables, divided into three categories. The categories are location tables, log tables and user tables.

10.1 Location tables

The location tables consist of five tables, as shown in Figure 10.1. The motivation for these tables is that one attraction can have several maps covering different territories, each map can cover several zones, each zone can contain several objects, whilst each object can be connected to several preferences. The preferences of an object are used with the dynamic menus service. The user can state his preferences in the attraction, and only objects matching his preferences will be displayed in the menu on his handheld device. In addition, there must exist a table with mapping information to transform the locations from real world coordinates to coordinates adjusted to fit on a map on the handheld device.

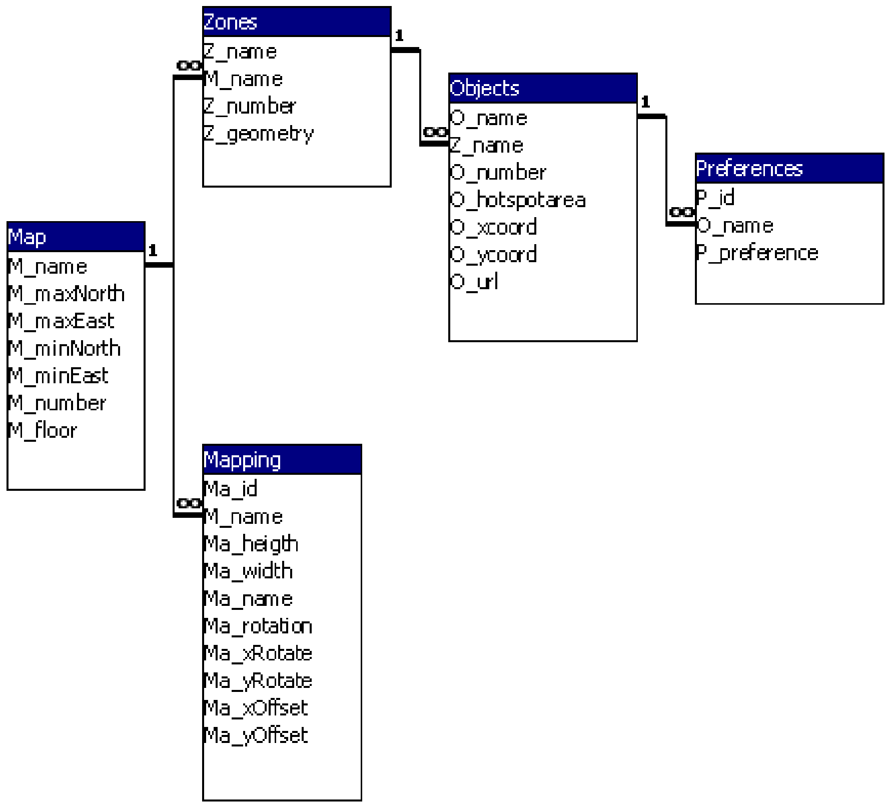


Figure 10.1: Location tables

As mentioned above, one attraction can have several maps covering different territories. Table 10.1 gives a short description of the different attributes in the *Map* table.

Field name	**Data type**	**Explanation**
M_name	varchar	The name of the map, primary key
M_maxNorth	integer	The maximum northing value of the map
M_maxEast	integer	The maximum easting value of the map
M_minNorth	integer	The minimum northing value of the map
M_minEast	integer	The minimum easting value of the map
M_number	integer	The number of the map
M_floor	integer	The floor the map is situated at

Table 10.1: Map table

Each map can contain several zones. Table 10.2 gives a short description of the different attributes in the *Zones* table.

Field name	**Data type**	**Explanation**
Z_name	varchar	The name of the zone, primary key
M_name	varchar	Foreign key of table Map
Z_number	integer	The number of the zone
Z_geometry	geometry	The circumference of the zone
Z_floor	integer	The floor the zone is located in

Table 10.2: Zones table

Each zone can contain several objects. Table 10.3 gives a short description of the different attributes in the *Objects* table.

Field name	**Data type**	**Explanation**
O_name	varchar	The name of the object, primary key
Z_name	varchar	Foreign key of table Zones
O_number	integer	The number of the object
O_hotspotarea	geometry	The hotspotarea covering the object
O_xcoord	integer	The x coordinate of the object
O_ycoord	integer	The y coordiante of the object

Table 10.3: Objects table

Each object can be connected to one or several preferences. The preferences are words like ”Mozart” or ”Beethoven”, and these preferences will be compared with the user’s preferences in the *UserPreferences* table. Table 10.4 gives a short description of the different attributes in the *Preferences* table.

Field name	**Data type**	**Explanation**
P_id	auto increment	Primary key
O_name	varchar	Foreign key of table Objects
P_preference	varchar	A keyword connected to the object

Table 10.4: Preferences table

To each map, some mapping information has to be available. This information is used to transform the position information from real world coordinates to coordinates adjusted to fit on the map on the handheld device. Table 10.5 gives a short description of the different attributes in the *Mapping* table.

Field name	**Data type**	**Explanation**
Ma_id	auto increment	Primary key
M_name	varchar	Foreign key of table Map
Ma_height	integer	Mobile device's(MD's) screen height in pixels
Ma_width	integer	MD's screen width in pixels
Ma_name	varchar	The name of the mapping
Ma_rotation	double	Radians to rotate MD's location to fit screen map
Ma_xRotate	double	The x coordinate of the transformation point
Ma_yRotate	double	The y coordinate of the transformation point
Ma_xOffset	integer	Pixels to move MD's location to fit screen map
Ma_yOffset	integer	Pixels to move MD's location to fit screen map

Table 10.5: Mapping table

10.2 User tables

The user tables consist of three tables, the *User* table, the *UserGroup* table and the *UserPreferences* table (Figure 10.2). Each user automatically recieves a user id. This id is used to separate the different users in the *LogLocation* table. If the user wants to use the ”locate friends” or ”send message” services, he has to register with additional information.

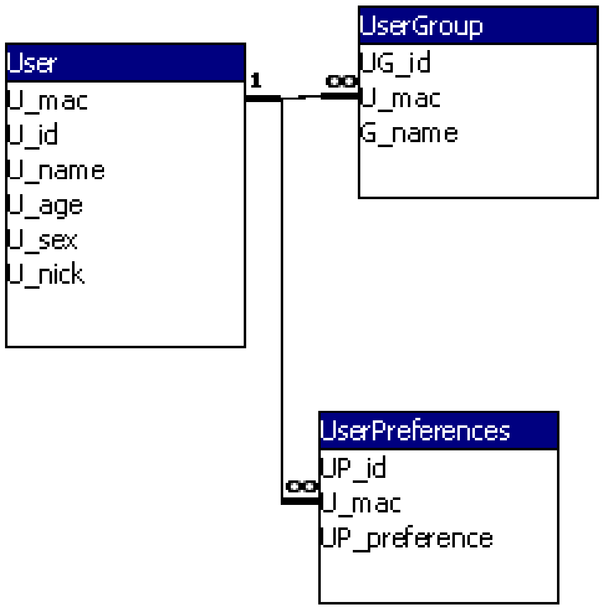


Figure 10.2: User tables

The nicks have to be unique so that only one person can use a nick at a time. After the user has terminated his session, the information about the user in the database can be deleted and other customers can use nicks that currently aren't in use.

The *User* table stores information about the different users of the Runtime System. Table 10.6 gives a short description of the different attributes in the *User* table.

Field name	**Data type**	**Explanation**
U_mac	varchar	Primary key
U_id	integer	An id to identify each user
U_name	varchar	The user's name
U_age	integer	The user's age
U_sex	varchar	The user's sex
U_nick	varchar	The user's nick

Table 10.6: User table

The *UserGroup* table consists of information about the people who are registered as a person's friend. This information is used to verify that one user is allowed to either state another user's location or send a message to another user. Table 10.7 gives a short description of the different attributes in the *UserGroup* table.

Field name	**Data type**	**Explanation**
UG_id	auto increment	Primary key
U_mac	varchar	The mac address of a person in the group
G_name	varchar	The name of the group

Table 10.7: UserGroup table

The *UserPreferences* table is used to store information about the user's main interests. By using this information, it is possible to give the user information adjusted to his interests. Table 10.8 gives a short description of the different attributes in the *User-Preferences* table.

Field name	**Data type**	**Explanation**
UP_id	auto increment	Primary key
U_mac	varchar	The user's mac address
UP_preference	varchar	A preference the user has

Table 10.8: UserPreferences table

10.3 Log tables

The log tables consist of two tables; one table for storing the users movements and one table for storing what kind of objects the users were interested in (Figure 10.3).

By using this information, it is possible to create various statistics about the users preferences.

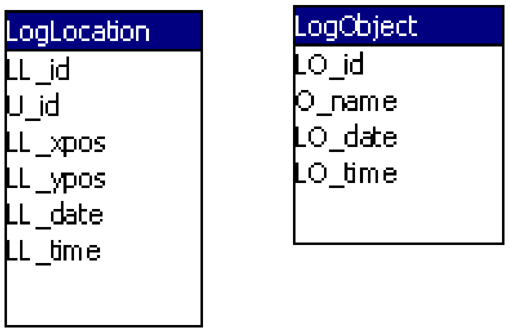


Figure 10.3: Log tables

The *LogLocation* table keeps track of the user's movement in the attraction area. Table 10.9 gives a short description of the attributes in the *LogLocation* table.

Field name	**Data type**	**Explanation**
LL_id	auto increment	Primary key
U_id	integer	The identification of the user
LL_xpos	integer	The x coordinate of the user
LL_ypos	integer	The y coordinate of the user
LL_date	date	The date the user was there
LL_time	time	The time the user was there

Table 10.9: LogLocation table

Each time a user views a presentation of an object, a row in the *LogObject* table is made. This information can be used to create statistics about the most popular or least popular objects in the attraction area. Table 10.10 gives a short description of the different attributes in the *LogObject* table.

Field name	**Data type**	**Explanation**
LO_id	auto increment	Primary key
O_id	integer	The identification of the object viewed
LO_date	date	The date the object was viewed
LO_time	time	The time the object was viewed

Table 10.10: LogObject table

Chapter 11

XML interfaces

Before starting to implement the system, several XML interfaces were made for handling the communication between the mobile devices and the Runtime System (RS). The client application can send a request to the RS in several different ways, but it has to follow a predefined syntax. The response will also follow a predefined syntax. This section presents the legal XML interfaces, and several examples are given. For a complete outline of the XML interfaces see Appendix E and Appendix F. Appendix E contains a DTD describing the XML transfer protocol while Appendix F presents several examples on legal XML strings.

11.1 XML elements in the request

The root element in every XML request must be a *locationRequest* as in the example below. This root element can contain several different elements depending on what kind of information that is wanted. These elements are presented in the following sections.

```
<locationRequest>....</locationRequest>
```

11.1.1 The *mac* element

Every *locationRequest* must contain an element called *mac*. The *mac* element holds the user's own mac address. The RS needs the mac to identify and find the position of each user. The *mac* element syntax is:

```
<mac>XX:YY:ZZ:XX:YY:ZZ</mac>
```

11.1.2 The *get* element

The *get* element is optional and is used when the client application wants to recieve information from the RS. Table 11.1 describes the different values the *get* request can have.

***get* - element**
position
simulatedPosition
friendsPosition
dynamicInfo
tracks
messages

Table 11.1: *get* commands

The following example shows the *get* element with the position command:

```
<get>position</get>
```

The <get>simulatedPosition</get> and the <get>tracks</get> commands are a bit different from the other *get* commands. In addition to only sending the command as in the *position* example over, it is possible to send extra information.

When using *simulatedPosition* the client application have to send information about the start and stop position. It also has to describe the step length. The following example shows a typical request. All values confer to the used positioning systems local coordinate system.

```
<locationRequest>
        <mac>00:01:43:EC:13:AB</mac>
        <get>simulatedPosition
               <startX>25</startX>
               <startY>50</startY>
               <stopX>50</stopX>
               <stopY>110</stopY>
               <stepX>1</stepX>
               <stepY>1</stepY>
        </get>
</locationRequest>
```

With the *track* command it is possible to send the *name* of the track that is wanted. In the following example, the client application asks for the track named *long*. If no specific name is given, the system will respond by sending all available tracks.

```
<locationRequest>
        <mac>00:0B:FD:C6:5C:AB</mac>
        <get>tracks
              <name>long</name>
        </get>
</locationRequest>
```

The <get>friendsPosition</get> request will by default return information about all the friends locations. In current version of the XML protocol the default value is the only option, but in the future versions it should be possible to locate one or several named friends.

11.1.3 The *mapping* element

The *mapping* element is necessary when the clients uses the <get>position</get> and the <get>friendsPosition</get> element. If it is omitted, a default mapping value is used. This default value is set by the Creator Tool when configuring the RS. The value of the mapping is the name stored in the database. The following example shows the syntax of the *mapping* element:

```
<mapping>PDA</mapping>
```

11.1.4 The *send* element

Like the *get* element the *send* element is also optional. *Send* is a bit more complex than *get*, and it contains a small amount of nested elements that is to be parsed by the RS. In current version of the XML protocol, the only send option available is sending messages. However, it is possible to extend the *send* element when new services are implemented. The following example shows the *send* element:

```
<locationRequest>
     <mac>00:0B:FD:C6:5C:AB</mac>
     <send>
         <message id="1">
               <to>truls</to>
               <header>Hallo</header>
               <body>How are you?</body>
         </message>
     </send>
</locationRequest>
```

11.2 XML elements in the response

The root element in every XML response must be a *locationResponse* as shown in the example below. This root element can contain several different elements depending on what kind of response that is required. If the requested information for some reason isn't available and elements in the response don't get any value, the server will give these elements the value ”noinfo” or ”-1” depending on wether the element contained a string or a number.

Every *get* element in the client request recieves a separate element in the response. These elements are presented in the following sections.

```
<locationResponse>....</locationResponse>
```

11.2.1 The *position* element

The *position* element is the response to the <get>position</get> request. The position element contains information about the user, which map the user belongs to and which zone and hotspot object the user resides in. The client application must then decide what to do with this information. The following example shows the syntax of a *position* element:

```
<locationResponse>
     <position>
          <user>
               <mac>00:0B:FD:C6:5C:AB</mac>
               <mapping>PDA</mapping>
               <x>100</x>
               <y>130</y>
          </user>
          <map>
               <name>The first map</name>
               <number>1</number>
               <floor>1</floor>
          </map>
          <zone>
               <name>Entrance</name>
               <number>3</number>
          </zone>
          <hotSpotObject>
               <name>Statue 1</name>
               <number>1</number>
               <x>100</x>
               <y>120</y>
          </hotSpotObject>
     </position>
</locationResponse>
```

In the cases where the client isn't located inside one of the predefined maps in the database, it is still possible that the Location Server returns a location on a <get>position</get> request. In this case, the following is the syntax of the response:

```
<locationResponse>
     <position>
          <user>
               <mac>00:0B:FD:C6:5C:AB</mac>
               <mapping>PDA</mapping>
               <x>100</x>
               <y>130</y>
          </user>
          <map>noinfo</map>
          <zone>noinfo</zone>
          <hotSpotObject>noinfo</hotSpotObject>
     </position>
</locationResponse>
```

11.2.2 The *friendPosition* element

The *friendPosition* element is the response to the <get>friendsPosition</get> request. The *friendPosition* element contains information about all the persons that are registered as the user's friends and their location. The following example shows the syntax of a *friendPosition* element:

```
<locationResponse>
     <friendPosition>
          <user>
               <nick>truls</nick>
               <x>100</x>
               <y>130</y>
               <map>
                    <name>Map 1</name>
                    <number>1</number>
               </map>
          </user>
          <user>
               <nick>fredrik</nick>
               <x>110</x>
               <y>125</y>
               <map>
                    <name>Map 1</name>
                    <number>1</number>
               </map>
          </user>
     </friendPosition>
</locationResponse>
```

If the client doesn't have any friends registered, the following is the syntax of the response:

```
<locationResponse>
     <friendPosition>noinfo</friendPosition>
</locationResponse>
```

11.2.3 The *dynamicInfo* element

The *dynamicInfo* element is the response to the <get>dynamicInfo</get> request. The *dynamicInfo* element will give information depending on predefined preferences. The *dynamicInfo* element contains information about all available *maps*, *zones* and *objects*. This information can be used to show dynamic menus. By using this service, it is possible to make changes to the location of objects, zones and maps both in the attraction area and in the database without having to make changes on the client application. The following example shows the syntax of a *dynamicInfo* element:

```
<locationRespons>
     <dynamicInfo>
          <map>
               <name>first</name>
               <number>1</number>
               <floor>1</floor>
               <zone>
                    <name>Entrance</name>
                    <number>1</number>
                    <object>
                         <name>Statue</name>
                         <number>1</number>
                         <x>100</x>
                         <y>120</y>
                         <hs>yes</hs>
                         <url>/image/statue3</url>
                    </object>
              </zone>
          </map>
     </dynamicInfo>
</locationRespons>
```

If no dynamic information is available, the following is the syntax of the response:

```
<locationResponse>
     <dynamicInfo>noinfo</dynamicInfo>
</locationResponse>
```

11.2.4 The *tracks* element

The *tracks* element is the response to the <get>tracks</get> request. This is a service the client can make use of to fetch some predefined tracks covering the attraction area. The response is a list containing coordinates that describe the track. It is up to the client to decide what to do with this information and how to present the track to the user. The following example shows the syntax of a *tracks* element:

```
<locationResponse>
     <tracks>
          <track>
               <name>long</name>
               <number>2</number>
               <coord>
                    <x1>324</x1>
                    <y1>232</y1>
                    <x2>324</x2>
                    <y2>232</y2>
                    <x3>324</x3>
                    <y3>232</y3>
                    <x4>324</x4>
                    <y4>232</y4>
                    <x5>324</x5>
                    <y5>232</y5>
                    <x6>324</x6>
                    <y6>232</y6>
              </coord>
          </track>
     </tracks>
</locationResponse>
```

If no tracks are available, the following is the syntax of the response:

```
<locationResponse>
     <tracks>noinfo</tracks>
</locationResponse>
```

11.2.5 The *message* element

The *message* element is the response to the <get>messages</get> request. The message elements contain all the unread messages to a user. The following example shows the syntax of a *message* element:

```
<locationResponse>
     <message>
          <note id="1">
               <to>me</to>
               <from>truls</from>
               <header>Hallo</header>
               <body>How are you?</body>
          </note>
          <note id="2">
               <to>me</to>
               <from>frank</from>
               <header>Hallo</header>
               <body>Do you want a break?</body>
          </note>
     </message>
</locationResponse>
```

If no messages is available, the following is the syntax of the response:

```
<locationResponse>
     <message>noinfo</message>
</locationResponse>
```

11.2.6 The *error* element

The *error* element is used by the RS to notify the client when something is wrong. An example is a request from the client application that are sent in an illegal syntax. The following example shows an example of an error message:

```
<locationResponse>
     <error>
          <type>unknown</type>
          <text>illegal syntax</text>
     </error>
</locationResponse>
```

Chapter 12

Runtime System

This chapter presents the implemented Runtime System (RS). The first section gives a general explanation of the system and the second section presents the main implementation details.

12.1 Explanation of the system

The RS is the main engine in the framework and is responsible for all functionality regarding positioning calculations. The system is responsible for transferring information about the tour attraction to the handheld devices adjusted to the user's location. It is therefore not necessary for the user to search through a lot of data to find information regarding sights he's close to.

12.1.1 Information flow

Figure 12.1 shows the main dataflow in the RS. The dataflow is initialized by a request to the RS, asking for information adjusted to the client's location. The RS finds the client's location and does the necessary calculations to find out which map and zone he is located in. In addition, the RS investigates whether the client is nearby a hotspot. When all the positioning information is calculated, the information is returned back to the client application on the mobile device.

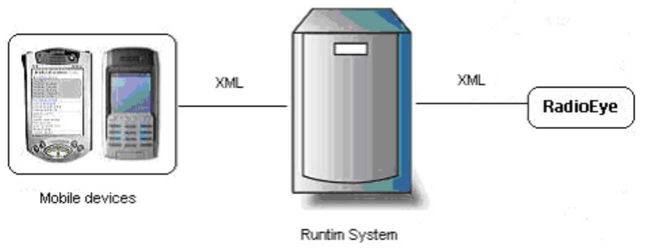


Figure 12.1: Runtime System

The current implementation of the RS does not use an external Location Server (LS) to manage the devices current locations as suggested in chapter 9 describing the architecture. Instead, the RS has direct contact with a positioning system called Cordis RadioEye, presented in section 2.3.1. This is done to reduce the development time of the prototype. However, it is recommended to use the LS in future versions.

12.2 Implementation details

The RS is implemented as a Java Servlet using J2SE. The system runs on a Tomcat Server and is connected to a MySQL database.

There has been developed a fully operational prototype of the RS that are tested with two different client applications. The client applications are developed for the Nidaros Cathedral, and are further presented in chapter 16. However, only four of the eight functional requirements for the RS are implemented in the prototype.

12.2.1 Class diagram

Figure 12.2 illustrates the complete class diagram for the implemented RS. The implementation follows the architecture, but since only parts of the framework are developed, not all classes are implemented. There are also made a few changes in the relations between some classes.

The *Constants* class is new and not part of the architecture. The class contains constants that are used during runtime. Examples on constants are *DB_URL*, *DB_USERNAME* and *DB_PASSWORD*. The *Constants* class my be exchanged by a *.properities* file that's read during start-up.

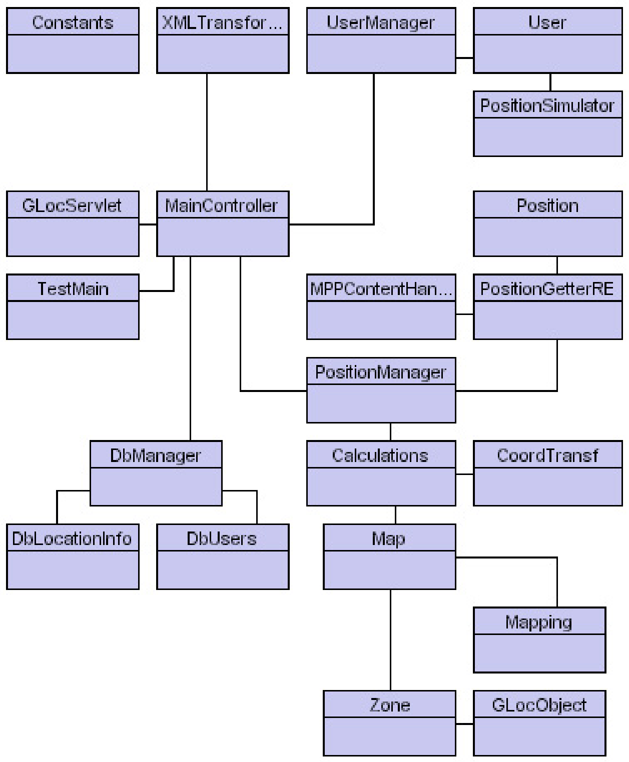


Figure 12.2: Implemented classes - Runtime System

12.2.2 Positioning

The *PositionGetter* class from the architecture has been replaced with *PositionGetterRE*. The *PositionGetterRE* class has a direct connection to the RadioEye, while the *PositionGetter* class has a connection to the LS.

The communication with the RadioEye is performed via XML, as illustrated in Figure 12.1. The RS and the RadioEye support different kinds of coordinate systems. However, the developed demonstrators use the RT-90 system, introduced in section 5. RT-90 uses local coordinates measured in meters.

The RadioEye localizes the users and returns their locations back to the RS. The locations are used to calculate which maps and zones the users are located in, and whether they are nearby any hotspots or not.

12.2.3 Database

All information about maps, zones and object are stored in a MySQL database. The same applies to the information about users and friends. Appendix G describes the creation of the database with tables. The Appendix also contains the data used during the Nidaros Cathedral demonstration.

During start-up, the RS connects to the database and stores the information in internal objects. A problem may occur with this approach if the database is larger than RAM. However, the database is supposed to handle the calculations itself in the future. The database is using a datatype called *Geometry*. There are supposed to to be some functions available to this data type, but these functions are not yet implemented in the MySQL database.

When a client application sends a request to the RS, the RS traverses its own objects and does the necessary calculations to produce a response. There is no need to make a new connection to the database at these requests. This reduces the response time to the RS. The total time used from a request is sent to the RS have performed the necessary calculations and returned a response, is measured to be less then 1 second.

12.2.4 Geometry calculations

Each *Map*, *Zone* and *GLocObject* object is, as stated in section 12.2.3, stored internally in the RS. The *Zone* and *GLocObject* use the class *Polygon* to store their coverage area. To find a users location, the *Calculation* class is used. First a search is performed through all the zones. When the correct zone is found, the RS also know which map the user is located in, because a zone is only located in one map. After the zone is determined, the *Calculation* class searches through all the *GLocObject* objects that are located inside the particular zone to find out whether the user is nearby a hotspot or not. The *GLocObject* class holds information about hotspots and implements the functional requirement ”F-5: Support hotspot functionality”.

12.2.5 Hysteresis

When a user is located in the borderline of a zone, the RS can send varying information about which zone the user is located in. The zone can change back and forth between the zone the user is located in and the zone he is nearby. The reason for this variation is the uncertainty in the location measurements from the RadioEye. To prevent this unwanted incident, a hysteresis is implemented.

The hysteresis extends the current zone with a predefined number of meters so that the user has to move a few meters out of the zone before he is told that he is located in a new zone. Likewise, when the user wants to enter the first zone again, he has to move the same number of meters into the first zone before he is told that he is back in the zone again.

12.2.6 Transformation of points

After the user’s position is found, the RS have to transform the position to fit the map on the user’s screen. The class *Coordtransformation* in Figure 12.2 is responsible for everything that concerns transformation from real-world coordinates to screen coordinates. To perform the transformation, the Java Class *AfflineTransform* is used. The *Mapping* object defines how the transformation is carried out, and the client applications have to decide which mapping to use. If the mapping element is omitted in the *locationRequest*, the RS will use its default *Mapping* object during the transformation.

Together with section 12.2.4, this section has described the functionality that confirms to the functional requirement ”F-4: Support map functionality”.

12.2.7 Position simulator

An important feature of the RS is the possibility to simulate users positions. The client applications have to send a *locationRequest* with *simulatedPosition*, as defined in section 11.1.2, to start the simulator. When the RS receives the request, it will be sent to the *PositionSimulator* class and the class will generate a simulated position. The RS then calculates the necessary location information the same way as if the location was received from the RadioEye.

By using the simulator service, a developer of a tour guide can ask the RS for position information during development of the client applications without being dependent on an external positioning system. This will result in reduced development time.

The *PositionSimulator* in Figure 12.2 is moved from being connected to the *Position-Manager* class in the architecture chapter, to being connected to the *User* class. The reason for this is that every user needs their own simulator independent of the others. Other ways, the simulation values will interfere with each other.

The *PostionSimulator* class implements the requirement ”F-3: Simulate runtime system”.

12.2.8 Locate friends

The last requirement implemented in the RS is ”F-6: Support a service to locate friends”. The database stores information about who is friend with whom. Each user has its own user objects. This object contains a table with all the friends the user is registered with. When the client application sends a *locationRequest* asking for *friend-Position*, the RS traverses the friends table and returns a *locationResponse* with all of the friends last known positions.

12.2.9 Requirements not implemented

The requirements ”F-7: Support a service to send messages”, ”F-8: Support dynamic menus”, ”F-9: Support predefined tracks” and ”F-10: Support a service to maintain a log” are not implemented.

12.2.10 XML

The class *XMLTransformation* is responsible for transforming the incoming XML requests to information that is understandable by the RS. The RS then uses this information for calculating the response to send back to the user. When all the calculations are finished, the *XMLTransformation* class transforms the information into XML data and returns the information to the user as a *locationResponse*.

The RS uses XML to communicate both with the client applications and the RadioEye. The reason for using XML is to follow a uniform standard so that all types of clients can be connected to the system. A negative effect by using XML is the parsing time. This is not a problem on the server side,. However, it can in some occasions slow down the client applications considerably. The PDA application tested at the Nidaros Cathedral uses between 1 and 2 seconds extra for parsing each incoming XML-string. The parsing of the XML is therefore a matter that can improve the performance of the system considerably.

12.2.11 Test class

During the development of the RS, a class called *TestMain*, illustrated in Figure 12.1, was implemented. This class was used during the unit testing throughout the development phase. The class is used as a virtual client application, but instead of going through the *GlocServlet* class it uses the *MainController* class directly.

Chapter 13

Creator Tool

This chapter presents the implemented Creator Tool. The first section describes the system, and the second section describes the main implementation details.

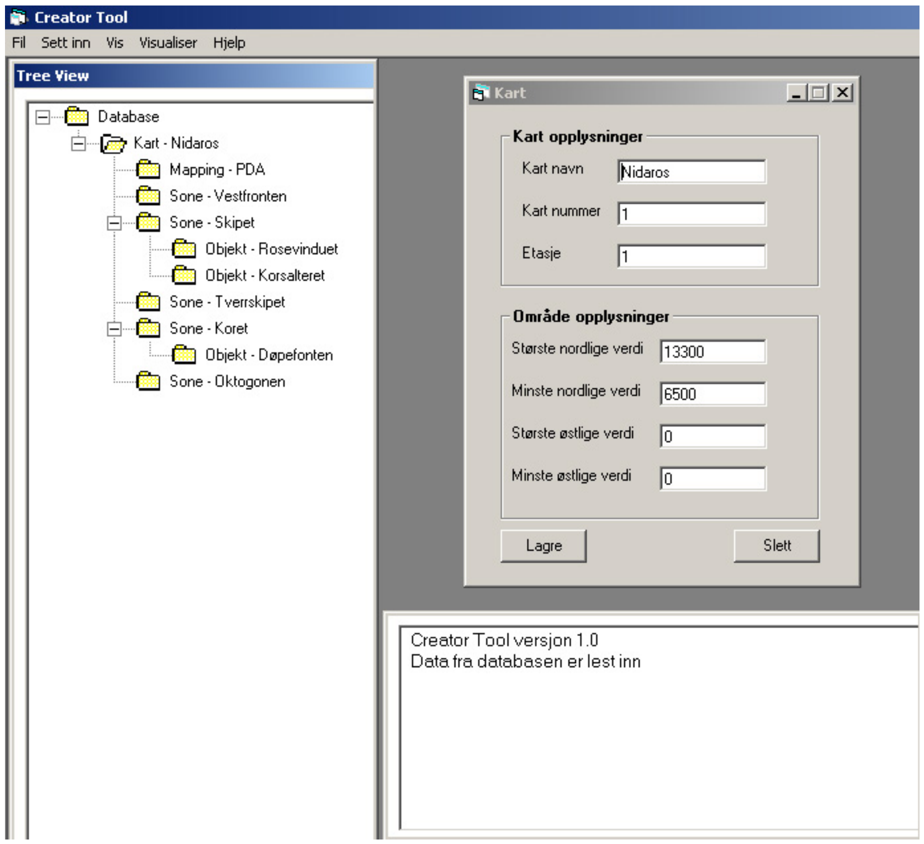
13.1 Explanation of the system

The Creator Tool is an application for registering information about new tour guides that are going to be developed, or editing an already existing tour guide. The user of the Creator Tool is the developer of a location-aware tour guide, and is expected to have good computer skills.

The Creator Tool is directly connected to a database where all location information about the specific attraction is stored. By using the Creator Tool, information in this database can be added, edited or deleted. The database used by the Creator Tool is the same as used by the Runtime System. Any changes made in the Creator Tool will therefore result in changes in the Runtime System.

13.1.1 Overview of the GUI

Figure 13.1 gives an overview of the graphical user interface of the Creator Tool. The various parts will be presented in the following sections.



Kart opplysninger

Kart navn	Nidaros
Kart nummer	1
Etasje	1

Område opplysninger

Største nordlige verdi	13300
Minste nordlige verdi	6500
Største østlige verdi	0
Minste østlige verdi	0

```
Creator Tool versjon 1.0
Data fra databasen er lest inn
```

Figure 13.1: GUI of the Creator Tool

13.1.2 The start-up window

The first window that appears during start-up, tells the developer to choose what kind of service he wants, Figure 13.2.

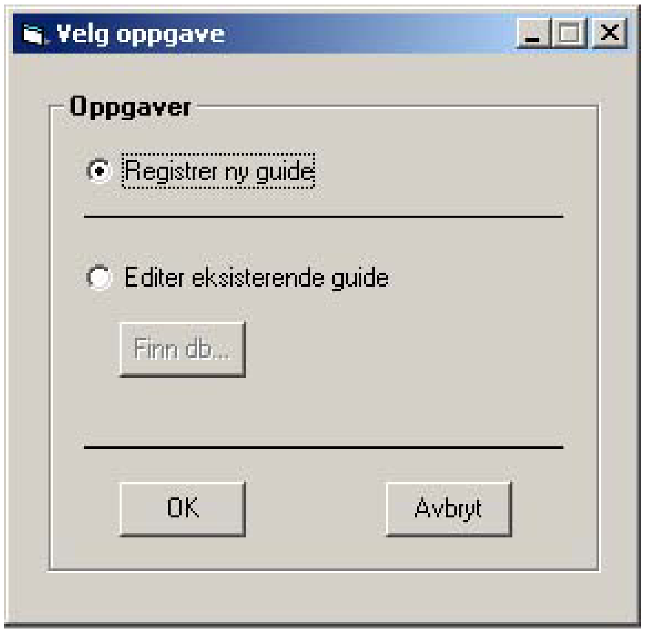


Figure 13.2: Start-up window

By default, the service to register a new guide is selected. If the user wants to edit an existing guide, he has to choose the ”Editer eksisterende guide” radio button. If this button is chosen, the ”Finn db” button is enabled. By pressing this button, a file containing the available databases appears.

13.1.3 The treeview structure

The graphical user interface for the Creator Tool is the same for the two options made during start-up. The only difference is that when you want to register a new guide, an empty treeview structure appears on the left side of the screen, while if you want to edit an existing guide, the treeview structure on the left is filled with information from the chosen database. A treeview structure of a filled tree is shown in Figure 13.3.

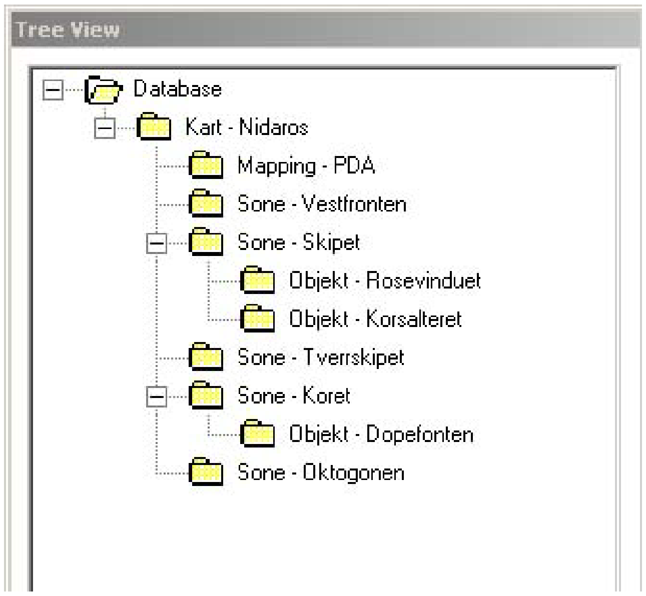


Figure 13.3: Treeview structure

The treeview structure is organized the same way as the database that is described in chapter 10. At the top level is the database. The next level in the treeview contains the maps. Each map can contain several mappings and zones. And finally, each zone can contain several objects.

The treeview is the most important structure in the Creator Tool. All the actions that take place in the tool are dependent on which node [1] that is chosen in the tree.

[1]A node is a uniform description for each element in the treeview

13.1.4 Insert new nodes

In the file menu ”Sett inn”, it is possible to add new nodes to the treeview structure. Based on what kind of node that is chosen in the tree, it is possible to add a new map, mapping, zone or object node.

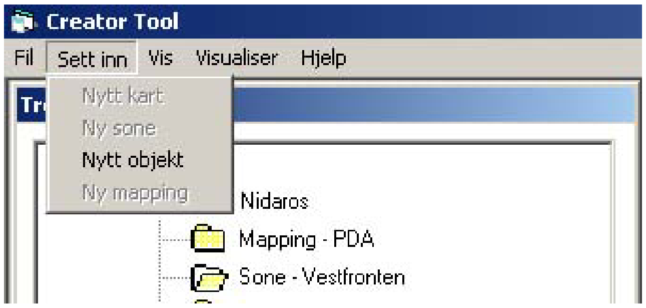


Figure 13.4: Insert new nodes

In Figure 13.4, the zone node ”Vestfronten” is selected. If a zone node is selected, the only menu item enabled in the ”Sett inn” menu is ”Nytt object”. If you choose to insert a new object, the object will be added at the level below the ”Vestfronten” zone. An object inserted here is located in the ”Vestfronten” zone. This is demonstrated in Figure 13.5, where ”Nytt objekt” is the object recently inserted, and correctly located in the level below ”Vestfronten”.

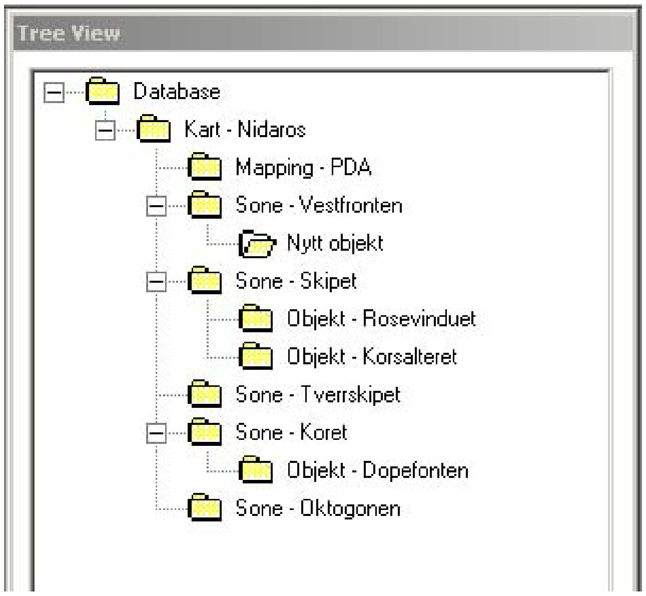


Figure 13.5: A new object node inserted

As explained above, the insertion of nodes in the treeview structure is dependent on which node that is selected in the tree:

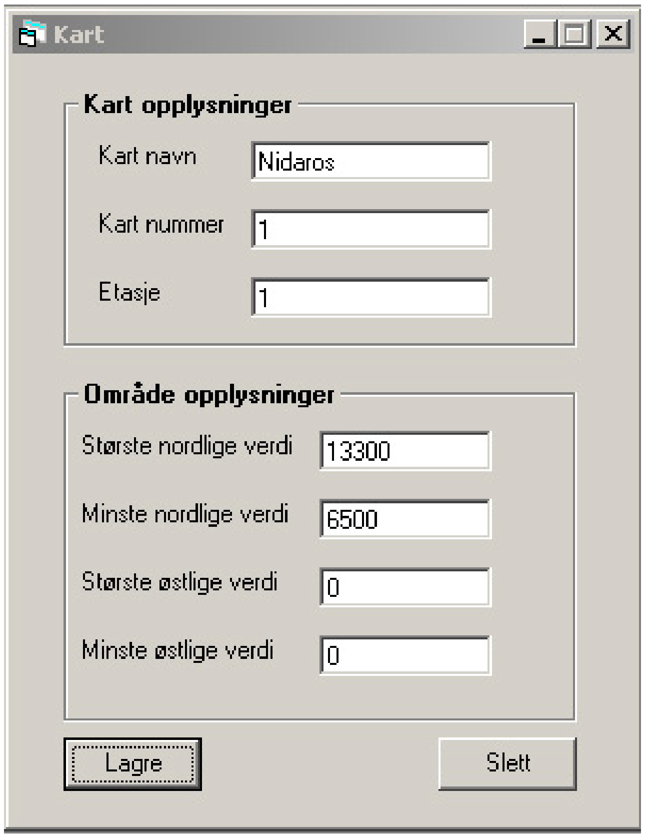
- If the database node is selected, it is possible to insert a map node
- If a map node is selected, it is possible to insert both mapping and zone nodes
- If a zone node is selected, it is possible to insert an object node
- If an object or mapping node is selected, it is not possible to insert any node

13.1.5 Displaying information

A click on a node in the treeview structure generates an event that opens a window on the right side of the treeview structure with information about the chosen node. The information in these windows corresponds to the data fields in the database, which is further described in the database chapter, chapter 10.

The map window

Figure 13.6 shows a map window filled with information. The map name is ”Nidaros”, it is map number one and it is located on the first floor. In addition, values about the area the map is covering are given.



Kart opplysninger

Kart navn	Nidaros
Kart nummer	1
Etasje	1

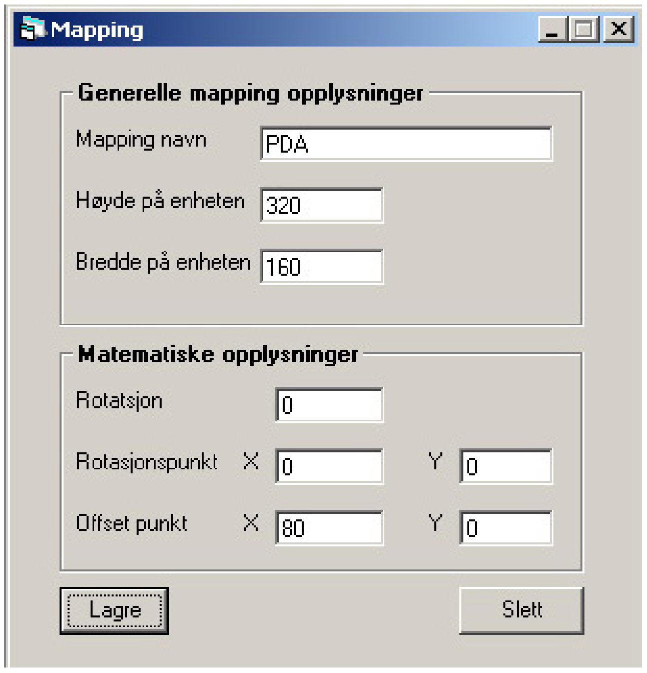
Område opplysninger

Største nordlige verdi	13300
Minste nordlige verdi	6500
Største østlige verdi	0
Minste østlige verdi	0

Figure 13.6: Map window

The mapping window

Figure 13.7 shows a mapping window filled with information. The mapping name is ”PDA” and the height and width of the device is given. In addition, values about the transformation from real world coordinates to map coordinates are given.



Mapping

Generelle mapping opplysninger

Mapping navn	PDA
Høyde på enheten	320
Bredde på enheten	160

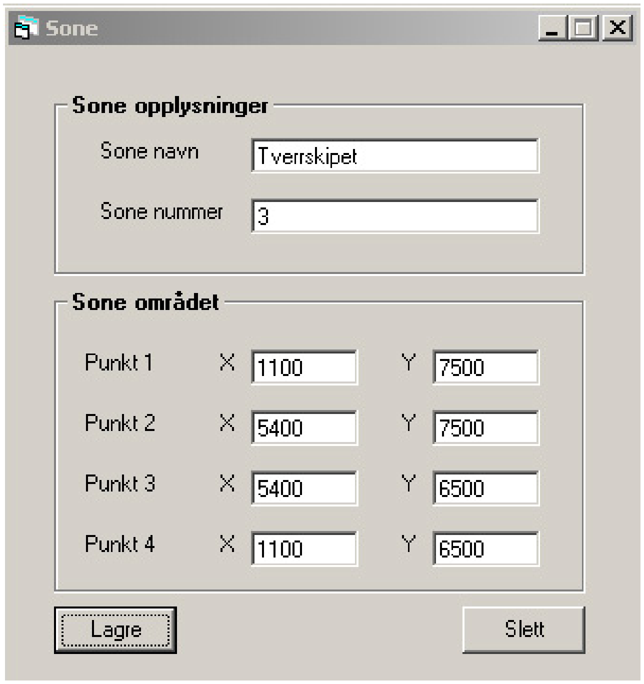
Matematiske opplysninger

Rotatsjon		0		
Rotasjonspunkt	X	0	Y	0
Offset punkt	X	80	Y	0

Figure 13.7: Mapping window

The zone window

Figure 13.8 shows a zone window filled with information. The zone name is ”Tverrskipet” and the zone number is three. Information about the zone area is also given.



Sone opplysninger

Sone navn: Tverrskipet

Sone nummer: 3

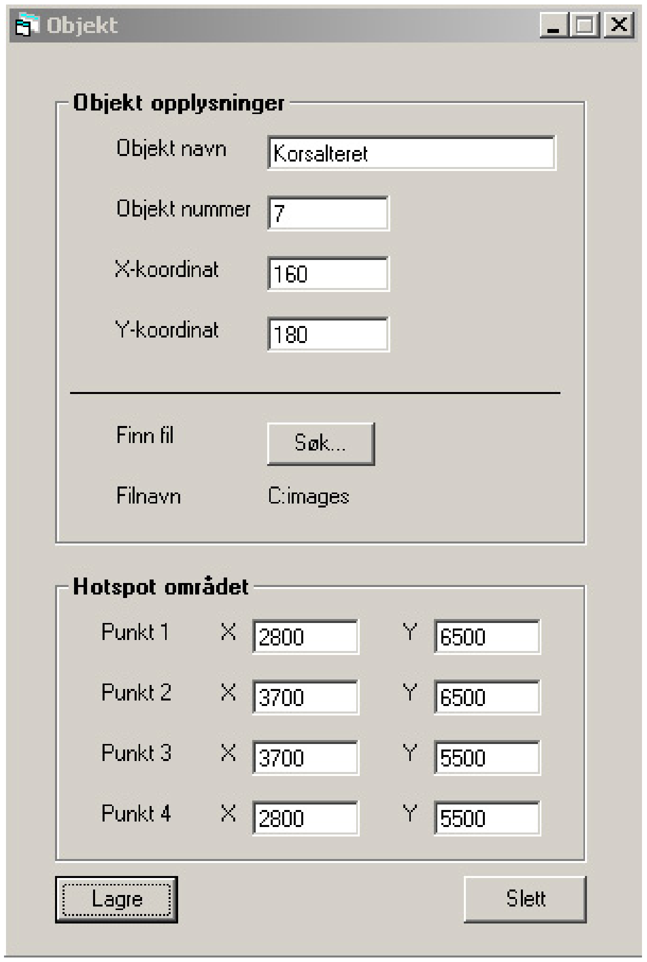
Sone området

Punkt 1	X	1100	Y	7500
Punkt 2	X	5400	Y	7500
Punkt 3	X	5400	Y	6500
Punkt 4	X	1100	Y	6500

Figure 13.8: Zone window

The object window

Figure 13.9 shows an object window filled with information. The name of the object is ”Korsalteret” and it is object number seven. The x-coordinate and y-coordinate is the pixels on the screen where the Korsalter shall be shown. The file name is the path to where the media presentation of the Korsalter is located. The last values describe a predefined area covering the Korsalter, the so-called hotspot area.



Objekt navn	Korsalteret
Objekt nummer	7
X-koordinat	160
Y-koordinat	180
Filnavn	C:images

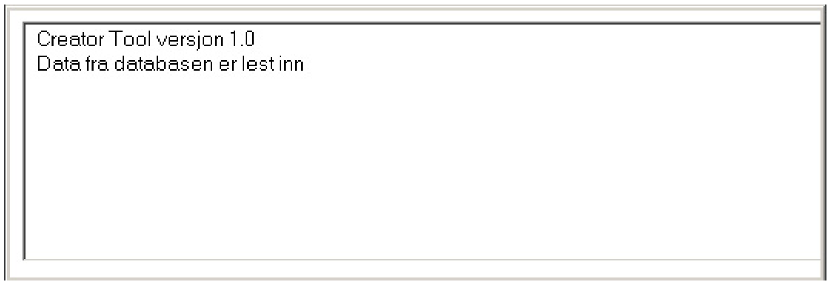
Punkt 1	X	2800	Y	6500
Punkt 2	X	3700	Y	6500
Punkt 3	X	3700	Y	5500
Punkt 4	X	2800	Y	5500

Figure 13.9: Object window

The windows presented in this section have two buttons in common, the ”Lagre” button and the ”Slett” button. By pressing the ”Lagre” button, the information in the chosen window is stored in the database, and by pressing the ”Slett” button; the information in the chosen window is deleted from the database. The node the window corresponds to is also removed from the treeview structure when the ”Slett” button is pressed.

13.1.6 The message window

A message window, shown in figure 13.10, is located at the bottom of the graphical user interface. All the messages generated from the system to the user during runtime are displayed in this window.



```
Creator Tool versjon 1.0
Data fra databasen er lest inn
```

Figure 13.10: Message window

13.1.7 Visualizing the guide

In the file menu there is a menu item ”Vis guide”, illustrated in Figur 13.11.

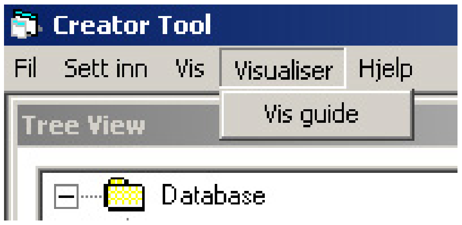


Figure 13.11: ”Vis guide” menu item

The ”Vis guide” service is supposed to draw a map according to the information given in the Creator Tool, with the zones and hotspot objects marked on it. This service is not implemented, instead a picture of what this map could look like is shown, Figure 13.12.

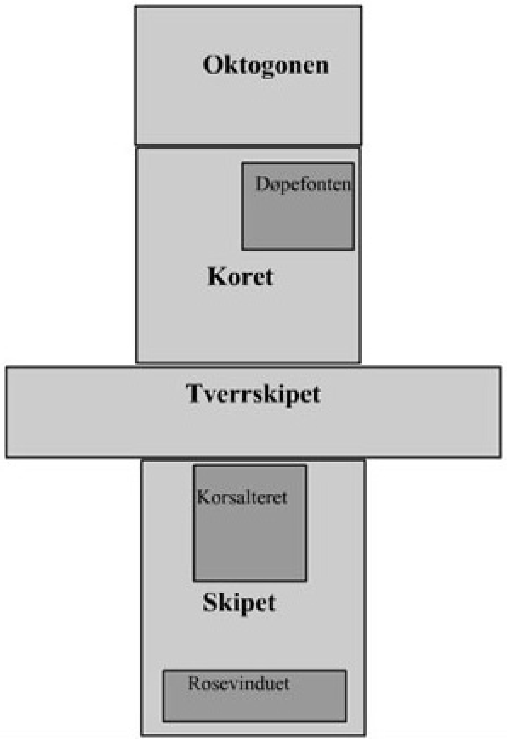


Figure 13.12: Visualization of the map

13.1.8 The File menu

In the file menu it is possible to make a new guide, open an existing guide or close the Creator Tool, illustrated in Figure 13.13.

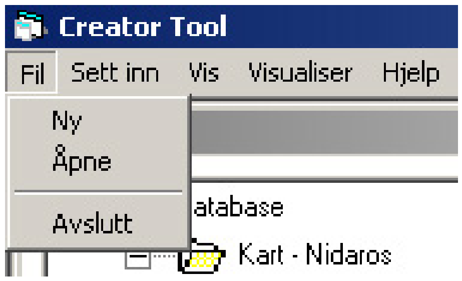


Figure 13.13: File menu

13.2 Implementation details

The Creator Tool is developed in Visual Basic (VB) by using the development tool Visual Studio 6.0.

The implementation of the Creator Tool is not complete, only a prototype is developed. The prototype demonstrates how this tool can be realized.

13.2.1 The database

To be able to communicate between the VB program and the database, a database driver is necessary. MySQL Connector/ODBC (MyODBC) driver is open source software and is used as the interface between the application and the database. MyODBC is one of the most popular ODBC Driver in the open source market, used to access the MySQL functionality [MyS].

The Creator Tool must have a connection to the same database as the Runtime System uses to be able to do any editing. In this version of the tool, the path to the database used by the Runtime System is written directly in the code. In a final version, the user of the tool must state the path to the database during runtime.

The requirement: ”F-1 Register a new tour guide” in section 8.2, is not completely fulfilled. It is possible to add the necessary information by using the Tool, and this information is stored in the treeview structure. However, none of this information is written to the database.

The requirement: ”F-2 Make changes to existing tourist guides”, is partially fulfilled. The application reads information from the database and stores it in the treeview structure. The limitation is on the editing of the information. Only changes that are made to object nodes will be stored in the database. Consequently, any changes made to map, mapping or zone nodes will not be registered in the database.

13.2.2 The treeview structure

As mentioned in section 13.1.3, the treeview is the most important structure of the Creator Tool. The treeview is a hierarchical list of related node objects. The user can expand and collapse these node families easily while navigating through the tree. Depicting the trees and expanding and contracting the nodes are handled automatically by the treeview itself.

Each node in the treeview structure has a unique index. When the user clicks on a node in the tree, this index is used to find out which node that is clicked and to display correct information. The index from the treeview is used as index in four separate arrays that stores the four different class types presented in section 13.2.5.

13.2.3 The forms

A form is where the graphical user interface is displayed. Each form in the Creator Tool is a separate window, and the tool consists of 11 forms.

13.2.4 Multiple Document Interface (MDI)

The Creator Tool is a MDI application. A MDI application is an application that can have multiple documents open at the same time, in contrast to Single Document Interface (SDI) where only one window can be open at the time.

A MDI application has one parent window (the application window), the rest of the windows are child windows. The parent window is always visible to the user. If the user closes this window, all other open windows are closed and the application terminates. Several child windows can be open at the same time, but a child window can never be moved outside the parent window.

In the Creator Tool, the parent window is the frame covering the application. The parent window contains the file menu. Two child windows are always visible in the parent window; the window containing the treeview structure and the window containing the messages to the users.

13.2.5 Class modules

In addition to the various forms used to make the graphical user interface, four different classes are used to store information about the maps, mappings, zones and objects during runtime. These classes consist of variables to store the information in the forms, and methods for retrieving these values.

13.2.6 The global module

There is one global module in the Creator Tool, and this module only contains two variables. The first variable stores the connection to the database. The last variable is a boolean variable that stores whether the system is used to edit an existing tour guide or to develop a new tour guide. If the system is used to edit an existing guide and the user wants to store a correction, the system has to alter the old information and do the corrections. If the system is used to develop a new tour guide and the user wants to store information, the system doesn't have to do any corrections before the information is stored.

Because these two variables are stored in a global module, they can be accessed from anywhere in the code.

Chapter 14

Statistical Tool

This chapter gives a presentation of the Statistical Tool. The Statistical Tool is not implemented, therefore only an explanation of the possible functionality of the system is given.

14.1 Explanation of the system

The Statistical Tool is for the staff of the attraction, and it shall be used to generate various statistics about the users' interests in the attraction.

During a guided tour, a user's actions shall be stored in a log. Each user receives a unique user id; hence no personal information is logged. Information that is interesting to log is the user's movements and what kind of sights the user was interested in. By using this information, the Statistical Tool can generate statistics. For instance, it can generate statistics about which sights were most or least popular.

14.2 Implementation details

Due to the time limitation of the thesis, the Statistical Tool is not implemented. Hence, no implementation details are given.

Chapter 15

Testing

This chapter covers the testing performed on the framework that is developed. The test procedures consisted of both unit testing and system testing, but the emphasis has been on the system testing. The system test is developed to reveal errors in the system compared with the requirement specification.

The Creator Tool, the Statistical Tool and the Runtime System are well documented in the architecture chapter. According to the project description, only parts of the framework were going to be implemented. Both the Runtime System and the Creator Tool are running applications, but not all the requirements are fulfilled. The Statistical Tool is not implemented.

15.1 Unit test

A unit test verifies that a function or module meets the requirements to the system. Unit tests include both black box testing and white box testing.

During development of the various tools, comprehensive black box testing has been performed continuous. The development was organized in modules. After a module was completed, it was tested with appropriate test data. The test data consisted of legal values, illegal values and marginal values. The unit testing is not documented any further, since the main focus of the testing has been on the system test.

15.2 System test

To perform the system test, several test scripts have been developed. One test script is only concerned with a specific part of the system, so that the various test scripts can run independently of each other. Some of the test scripts had to carried out more than one time. This was because the first test didn't give satisfactory results, and some improvements had to be made. In the following sections, only the final test scripts are included.

The system test is divided into two parts, the testing of the functional requirements and the testing of the non-functional requirements. The requirements are described in chapter 8.

The testing of the Creator Tool was accomplished by using information stored in the database tailored to the Nidaros Cathedral application. However, the test scripts for the Creator Tool are general. Therefore, no changes in the test scripts would be necessary to test the Creator Tool with another set of test data.

The testing of the functional requirements to the Runtime System has been accomplished by using the PDA application and cellphone application developed and tailored to the Nidaros Cathedral. However, it is important to emphasize that it is the Runtime System that is tested and not the client applications. By using the Nidaros Cathedral applications as test cases, it was possible to test the framework in a real-life environment. The testing of the Runtime System could also have been accomplished by sending various XML requests to the server, and then verify that the responses from the server were correct, both regarding the XML syntax in the response and the information in the XML.

The Runtime System was tested in the Nidaros Cathedral during preparation of the system before the demonstration on may 14th. During the testing, Cordis RadioEye, introduced in section 2.3.1, was used to localize the various users.

15.2.1 Testing of functional requirements

Creator Tool

F-1: Register new tourist guide
The developer shall be able to use the Creator Tool to register a new tourist guide, and store the necessary data in a database. Table 15.1 shows the testscript ”Register new tourist guide”.

Nr	What	How	Result	OK/ Error
1	**Register new tourist guide**			
1.1		Start the Creator Tool	The Creator Tool starts as an window application	OK
1.2		Make sure that the ”Registrer ny guide” radio button is selected	The ”Registrer ny guide” radio button is selected	OK
1.3		Press the OK button	No information is stored in the treeview on the left side	OK
1.4		Make sure that the ”database” node is selected in the treeview	The ”database” node is selected	OK

Nr	**What**	**How**	**Result**	**OK/ Error**
1.5		Press ”Sett inn” from the menu	The only menu item enabled in the ”Sett inn” menu is ”Nytt kart”	OK
1.6		Press ”Sett inn → Nytt kart” from the menu	A new ”Kart” window appears in the grey area on the right side, and the map is added to the treeview at the level below the database node with the caption ”Nytt kart”	OK
1.7		Make sure that the textboxes in the ”Kart” window are empty	The textboxes are empty	OK
1.8		Add appropriate text in the textboxes, and press the ”Lagre” button	The text ”Kartet *kartnavn* er lagret” appears in the messages window, and the name of the node in the treeview changes from ”Nytt kart” to ”Kart - *kartnavn*”. The data is also stored in the database.	OK*
1.9		Add a new Map by first selecting the database node in the treeview and then press ”Sett inn → Nytt kart” from the menu	A new ”Kart” window appears in the grey area on the right side, and the map is added to the treeview at the level below the database node with the caption ”Nytt kart”	OK
1.10		Add some incorrect information in the different text boxes, and press the ”Lagre” button	An appropriate and understandable error message appears. No information is stored in the database	Error
1.11		Repeat 1.6 to 1.10 for the other ”Sett inn” menu items		Error
1.12		Press two different nodes in the treeview, and verify that the correct window appears in the grey area to the right with correct information in the various text boxes	The correct window appears in the grey area with the same information as you typed in the various text boxes	OK
1.13		Choose a node in the treeview, then press the ”Slett” button in the window	A confirmation message appears, and when pressing OK the node and all its children are deleted from the treeview	OK*

Nr	What	How	Result	OK/ Error
1.14		Press ”Vis → Samlet vindu” from the menu	A new window appears with all the information stored in the treeview available from this window. All other windows are closed	OK
1.15		Make sure that correct information appears in the window when clicking on nodes in the treeview	Correct information appears in the window	Error
1.16		Press ”Vis → Enkelt vinduer” from the menu	The ”Samlet vindu” is closed	OK
1.17		Press ”Visualiser → vis guide” from the menu	A map is drawn that shows the zones and objects and their relationship	Error
1.18		Press ”Hjelp → Creator Tool hjelp” from the menu	A new window with help information appears	Error
1.19		Press ”Hjelp → Om Creator Tool” from the menu	A new window with general information about the Creator Tool appears	Error
1.20		Press ”Fil → Avslutt” from the menu	The Creator Tool is closed	OK
1.21		Start the Creator Tool again, and press the cross in the upper right corner to end the application	The Creator Tool is closed	OK

Table 15.1: Testscript ”Register new tourist guide”

OK*
In current version of the Creator Tool, no information is written to the database during development of a new tourist guide. In addition, it is only possible to delete object nodes from the treeview and not zone-, map- or mapping nodes.

Error in row 1.10, 1.15, 1.18 and 1.19
Not implemented

Error in row 1.17
The map is not generated based on the input from the user. The map is drawn in advance and corresponds to information that is adjusted to Nidaros Cathedral.

F-2: Make changes to existing tourist guide

The developer shall be able to make changes to existing guides. Table 15.2 shows the testscript ”Make changes to existing tourist guide”.

Nr	**What**	**How**	**Result**	**OK/ Error**
2	**Make changes to existing tourist guide**			
2.1		Start the Creator Tool	The Creator Tool starts as an window application	OK
2.2		Select the ”Editer eksisterende guide” radio button	The ”Finn db” button is enabled	OK
2.3		Press the ”Finn db” button	A file browser appears	OK
2.4		Find the correct database in the file browser and press the ”Open” button	The text ”Databasenavn: *databasenavn*” is visible	OK
2.5		Press the OK button	The treeview on the left side is filled with correct information from the database	OK (See comment)
2.6		Click on some nodes in the tree	Correct window appears with corresponding information	OK
2.7		Click on an object node in the tree. Make some changes to the data in the textboxes and press the ”Lagre” button	The new information is stored in the database and the message ”Objektet *objektnavn* er lagret” appears in the message field	OK
2.8		Click on a zone node in the tree. Make some changes to the data in the textboxes and press the ”Lagre” button	The new information is stored in the database and the message ”Sonen *sonenavn* er lagret” appears in the message field	OK*
2.9		Click on a map node in the tree. Make some changes to the data in the textboxes and press the ”Lagre” button	The new information is stored in the database and the message ”Kartet *kartnavn* er lagret” appears in the message field	OK*
2.10		Click on a mapping node in the tree. Make some changes to the data in the textboxes and press the ”Lagre” button	The new information is stored in the database and the message ”Mappingen *mappingnavn* er lagret” appears in the message field	OK*

Nr	**What**	**How**	**Result**	**OK/ Error**
2.11		Add a new Object by first selecting a zone node in the treeview and then press ”Sett inn → Nytt Objekt” from the menu	A new ”Objekt” window appears in the grey area on the right side, and the object is added to the treeview at the level below the zone node with the caption ”Nytt objekt”	OK
2.12		Add appropriate text in the textboxes, and press the ”Lagre” button	The text ”Objektet objektnavn er lagret” appears in the messages window, and the name of the node in the treeview changes from ”Nytt objekt” to ”Objekt - *objektnavn*”. The data is also stored in the database	OK
2.13		Repeat 1.6 to 1.10 for the other ”Sett inn” menu items		OK*

Table 15.2: Testscript ”Make changes to existing tourist guide”

OK*
Only changes in the information about objects are stored in the database. Changes in zones, maps or mapping are only temporarily stored in memory and are not written to the database.

Comment to row 2.5
The database that is used is predefined in the code, so the choices that are made in the file browser are of no importance.

The Runtime System

As mentioned in the beginning of this chapter, the Runtime System has been tested through the PDA application and the cellphone application tailored to the Nidaros Cathedral. These applications send requests to the server in correct XML syntax; parse the responses from the server and use the received information to display the results from the requests on the handheld devices.

F-3: Simulate Runtime System
There shall be a simulation service available that returns simulated location data to the client.

A client application that uses the simulation service is developed. This application is almost exactly the same as real client applications, except that the user's position is simulated. This simulation application was showed during the demonstration on May 15th, and no errors were found.

F-4: Support map functionality
If the clients have a map available, the framework shall provide the clients their positions in correct map coordinates and give information about which area the clients are located in. It shall also be possible to show the path the clients have covered on the map. Table 15.3 shows the testscript ”Support map functionality”.

Nr	**What**	**How**	**Result**	**OK/ Error**
3	**Support map functionality**			
3.1		Start the application on the PDA and choose ”Norsk” as language	A map covering the area appears	OK
3.2		Verify that there is a red circle on the map with your name on, and that this circle is situated at the correct place on the map according to your location	The red circle is situated in the correct location on the map	OK
3.3		Verify that the zone you are located in is emphasized on the map	The zone is emphasized	OK
3.4		Move around within the same zone	The red circle is moving on the map according to your movements. The zone you are located in is always emphasized.	OK*
3.5		Move to another zone on the map	The new zone is emphasized on the map, whilst the old zone is not	OK

Nr	What	How	Result	OK/ Error
3.6		Mark the trace check box	A line appears on the map showing your movements	OK

Table 15.3: Testscript ”Support map functionality”

OK*
There were some problems with inaccuracy and late response from the RadioEye during testing. There has been a close cooperation with employees from Radionor, and they made improvements to the RadioEye continuous (both in the RadioEyes code and configuration). Since the architecture of the Runtime System make use of an external location server, and not only the RadioEye, this problem is out of scope for this master thesis.

F-5: Support hotspot information
The framework shall be able to make the clients aware that a hotspot is nearby. Table 15.4 shows the testscript ”Support hotspot information”.

Nr	What	How	Result	OK/ Error
4	**Support hotspot information**			
4.1		Start the application on the PDA and choose ”Norsk” as language	A map covering the area appears	OK
4.2		Verify that the hotspot check box is not selected	The hotspot checkbox is not selected	OK
4.3		Move to a nearby hotspot object	Text with the name of the hotspot appears on the map where the hotspot is situated.	OK
4.4		Move out of the predefined area covering the object	The hotspot text is removed from the map	OK
4.5		Select the hotspot checkbox to automatic start a presentation when you are inside a predefined hotspot area	The hotspot checkbox is selected	OK
4.6		Move to a nearby hotspot object	The presentation of the hotspot object starts automatically	OK

Table 15.4: Testscript ”Support hotspot information”

F-6: Support a service to locate friends
It shall be possible to locate friends on the map. To use this service, a group of people

have to be registered as friends in the database. Table 15.5 shows the testscript ”Locate friends”.

Nr	**What**	**How**	**Result**	**OK/ Error**
5	**Locate friends**			
5.1		Start the application on the PDA and choose ”Norsk” as language	A map covering the area appears	OK
5.2		Verify that the ”Venner” checkbox is not selected	The checkbox is not selected	OK
5.3		Verify that there is a red circle on the map with your name on it, and that this circle is situated the correct place on the map according to your location	The red circle with your name is situated in the correct location on the map. No friends are located on the map	OK
5.4		Verify that you are registered with at least one friend in the database		OK*
5.5		Select the ”Venner” checkbox	The location of your friends who are situated on the area are stated on your map	OK
5.6		Verify that your friends movements are updated on your map	Their new locations are drawn on the map	OK

Table 15.5: Testscript ”Locate friends”

OK*
There is not implemented a service to register any friends in the database. Therefore, information about friends has to be typed directly into the database.

Requirement F-7 to F-10
The requirements F-7 (Support a service to send messages), F-8 (Support dynamic menus), F-9 (Support predefined tracks) and F-10 (Support a service to maintain a log) are not implemented. Therefore, no test scripts are developed for these requirements.

The Statistical Tool

The Statistical Tool is not implemented. Hence, no test scripts are developed for the requirements that apply to this tool.

15.2.2 Testing of the non-functional requirements

The non-functional requirements are more difficult to test than the functional requirements. Therefore, some of the non-functional requirements have been evaluated and not tested. The reason is that the testing of many of the non-functional requirements is too demanding to go carry out within a reasonable time limit.

NF-1: Usability
Experienced computer users shall be able to configure the Creator Tool only by using the support in the systems own documentation.

Evaluation No user documentation is made. Hence, this requirement could not be tested.

NF-2: Usability
The Runtime System shall always return understandable information, both when the client application states correct and incorrect input.

Evaluation
It is not implemented any error handling regarding whether the request from the client is correct or not. If the request is illegal, there will be no response at all from the server to the client.

NF-3: Portability
It shall be possible to extend the framework to run on newer versions of Windows than Window 2000.

Test
During development, both Windows 2000 and Windows XP have been used. No problems encountered.

NF-4: Extendibility
It shall be possible to add new positioning systems.

Evaluation
When the architecture was made, the possibility to add new positioning systems was of great importance. In the architecture, an external Location Server is used to receive the users location. It is possible to add any number of positioning systems to this Location Server if desired.

NF-5: Extendibility
There shall be possible to use different transmission protocols.

Test
The system has been tested with two different transmission protocols, namely WLAN

and GPRS. The guide that runs on the PDA uses a WLAN to transmit data from server to client, while the WAP application on the cell phone uses GPRS. No problems encountered with any of the transmission protocols.

NF-6: Performance
There shall not take more than 10 seconds before the Runtime System returns a response to a request.

Test
During the preparation and testing in the Nidaros Cathedral, the time was measured from a request was sent from the client until a response arrived at the client. The results showed that this process took less than two seconds.

NF-7 Maintainability
The framework shall be module based.

Evaluation
The framework consists of three separate tools, the Creator Tool, the Statistical Tool and the Runtime System. The framework is module based.

NF-8 Financial and time saving
The time used to develop a location-aware guide shall be less when using the framework instead of developing the guide from scratch.

Evaluation
Since the Runtime System is complete and ready to use, the work that has to be done on the server side to develop a location-aware guide is substantially reduced. In addition, by using the Creator Tool it is easy to configure the Runtime System to fit any attraction. Therefore, it is both timesaving and cost reducing to use the framework instead of developing a location-aware guide from scratch.

NF-9: Installability
It shall take less than 2 days to install and configure the Runtime System.

Test
On April 28th, necessary equipment was set up in Nidaros Cathedral to test the location-aware guides that were developed on both PDA and cellphone. The Runtime System was configured in the church in approximately 5 hours.

NF-10: Security
The stakeholders should not be able to enter any parts of the system they are not allowed to enter.

Evaluation

The Creator Tool is supposed to be used by developers, and by using this tool you have permission to change information that is stored in the database. The Statistical Tool, which is not implemented, is supposed to be used by the staffs. If this tool was implemented, the tool would only have permission to read from the database. Therefore, the staffs would not be able to do changes to any information that is stored in the database.

15.2.3 Summing up the system test

The last run of the test scripts gave good results and there were no unexpected errors. Most of the errors found were caused by the fact that the required services weren't implemented. According to the master thesis description, only parts of the framework were going to be implemented. Therefore, the lacking of these services is not considered significant. The implementation of the entire framework would constitute a much bigger task than it is possible to accomplish within the given time limit.

The testing of the non-functional requirements was difficult to perform. Some of the non-functional requirements are not tested satisfactory, only an evaluation of the requirement has been accomplished. However, the evaluations that are stated are thoroughly considered and should not contain any mistakes.

Part IV

Demonstrator

This part presents the client applications developed to demonstrate the framework's functionality. It presents two client applications, one for PDAs and the other for cellphones. It also includes a chapter describing the demonstration of the framework.

Chapter 16

Client applications

Two applications are made to test the functionality of the generic location-aware framework. The first for PDAs and the second for cellphones. Both applications are electronically location-aware tour guides made for the Nidaros Cathedral in Trondheim and are based on a guide implemented by Klipp og Lim Media.

This chapter explains how each system is constructed and provides examples of the visual appearance of the GUIs and the functionalities.

16.1 PDA application

The PDA application, also called Nidaros Pocket Guide (NPG), was coded in Flash MX. One of the reasons for using Flash is that it becomes easy to port from one operating system to another. Flash also runs smoothly compared with for instance Java because it's using less CPU.

The NGP application has been tested with Windows CE on 3 different versions of IPAQs and runs without errors.

16.1.1 Information flow

The NPG is directly connected to the Runtime System (RS), as shown in the component diagram in Figure 16.1. All communication between the two systems are done by the http protocol using WLAN.

The Flash client starts the information flow by sending a request to the RS. The request uses the XML syntax defined in chapter 11. The RS finds the position of the PDA by using the Cordis RadioEye. The RadioEye uses the PDA's WLAN signals to locate it. An extended description of the RadioEye is provided in section 2.3.1.

When the RS has located the PDA, it sends a XML response containing the location information back to the client application.

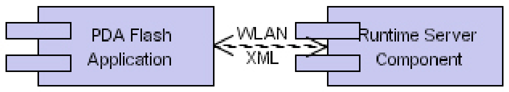


Figure 16.1: Component diagram - PDA application

16.1.2 User interaction

This section explains how the interaction between a user and the GUI is organized with respect to the GUIs. First there is an introduction on how a Flash application is built. This will make it easier to understand how the NPG is made.

A Flash application is made like a movie. It contains several frames and each frame represents a GUI. When the application starts, frame number one is shown. Then trough user input, it is possible to move to another frame. Every frame will be loaded at startup, which makes the jumping from a frame to another very smooth. The state diagram in Figure 16.2 shows the main frames in the NPG. These frames correspond to the following screenshots.

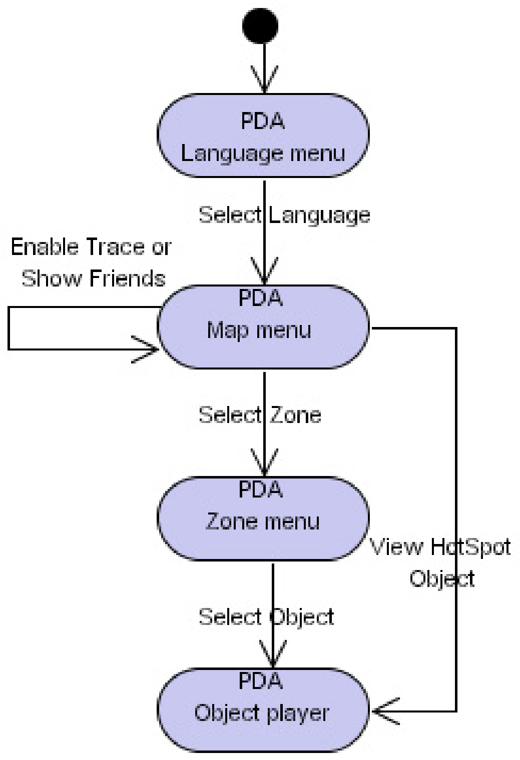


Figure 16.2: State diagram - PDA application

The state diagram shows the main features and is therefore not complete. However, the state diagram shows enough to explain the main flow in the program.

Language menu

The application starts with a language menu where the user can choose between three languages, illustrated in Figure 16.3.

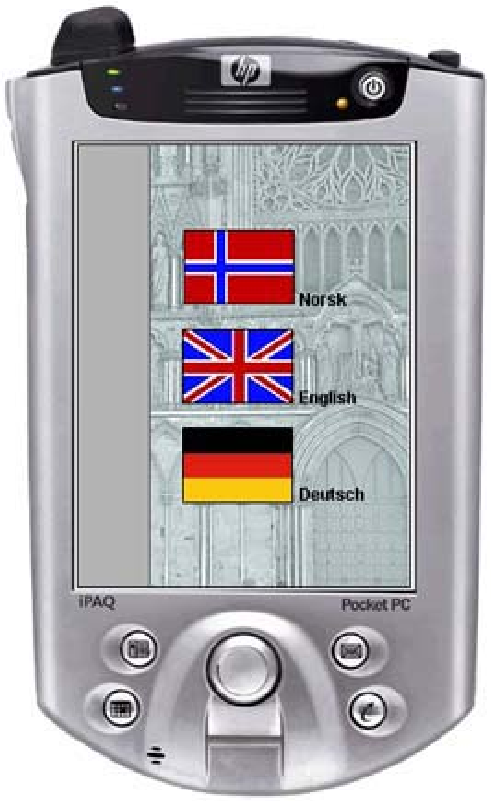


Figure 16.3: PDA application - Start

Map menu

The map menu contains the main features in the NPG application. As illustrated by the red circle in Figure 16.4, users are displayed on the map according to their location. The zone where the user is located is also emphasized by a red color surrounding it. Objects (hotspots) close to the user are displayed as a gray circle together with their name, illustrated in Figure 16.4 as ”Døpefonten”

In addition to the information regarding the users own positions, it is also possible to find out where your friends are located. All friends are shown with blue circles and names. To enable this service, you have to mark the check box in the upper left corner named ”Venner”.

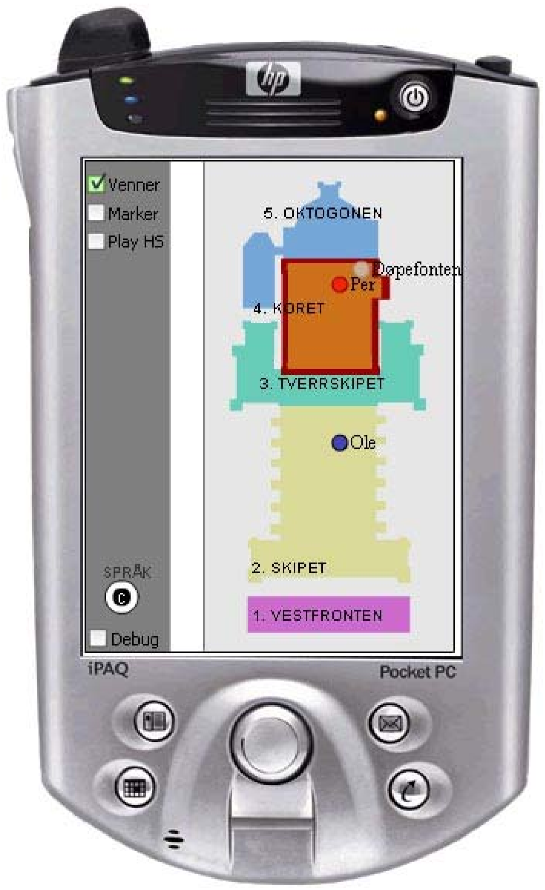


Figure 16.4: PDA application - Map

The remaining two check boxes in the upper left corner make it possible to either trace the movements you make or play a movie about a hotspot you are nearby.

Zone menu

The zone menu presents pictures of objects located in the zone, as illustrated in Figure 16.5. This menu is available by clicking a zone in the map screen.

If the user clicks on one of the pictures in the zone menu, the object player will appear on the screen.

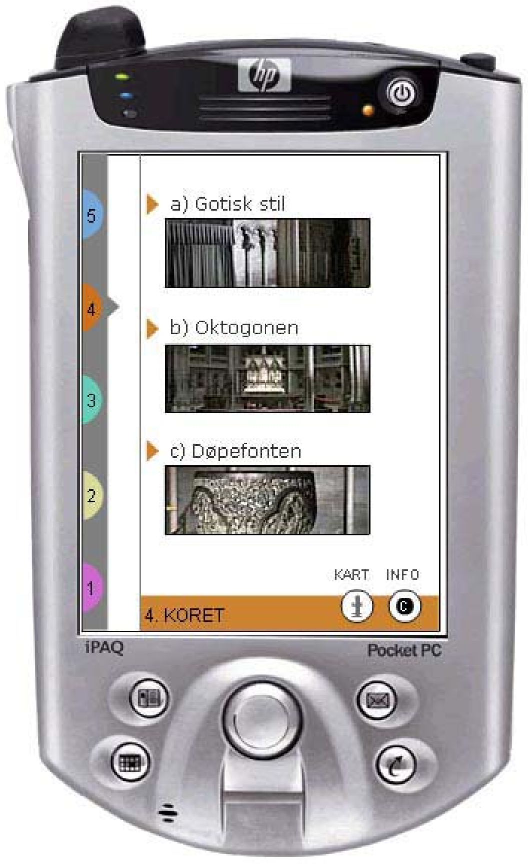


Figure 16.5: PDA application - Zone

Object player

The object player will give the user information about specific sights trough sound and pictures, lasting between 0.5-2.0 minutes. Figure 16.6 illustrates the object player. The player contains all the elementary functionalities that are needed to view video clips.

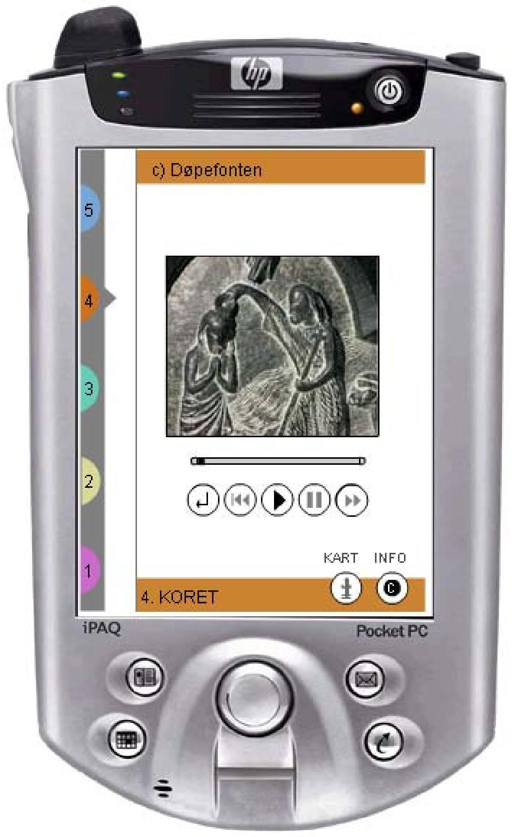


Figure 16.6: PDA application - Object

16.2 Cellphone application

The cellphone application, developed in cooperation with Telenor R&D, presents map and zone information in a web browser by using HTML, while the object movies are shown through the default media player installed on the cellphone. HTML is used to make the application easy to configure for new cellphones.

At the time this was written, the only cellphone capable of running the application is Eriksson's P900. However, there are continually being released new cellphone models, and within a short period of time it is expected to run on a number of models.

16.2.1 Information flow

In order to make the content on the cellphone as dynamically as possible, there are not installed any extra systems on the phone. Instead an external Java application is made, named ”Java Phone Component” in Figure 16.8, that continually asks the RS for the position of a tag the user is carrying.

The Cordis RadioEye is a WLAN positioning system. However, the cellphone communicates over the GPRS network and not through WLAN. It is expected that future phones will have WLAN support, but in this version the users have to wear an extra tag that transmits WLAN signals. Figure 16.7 illustrates the WLAN tag used in this project.

Figure 16.7: WLAN tag

The RS will find the position of the WLAN tag and return the position information to the Java Phone Component. The Phone Component then sends information to a gateway that forwards a SMS to the cellphone and tells it to download information from a web server. Figure 16.8 illustrates a simplified figure of the main components and dataflow in the system.

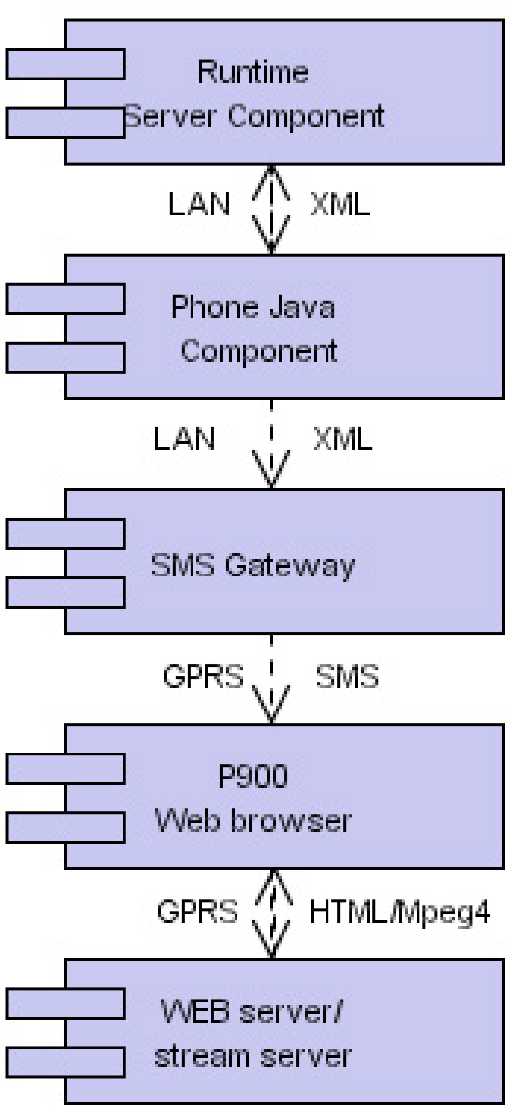


Figure 16.8: Component diagram - Cellphone application

16.2.2 User interfaces

The cellphone application does not require much user input, thus this section will briefly explain how the system is influenced by the users positions and automatic updates the GUIs on the phone. Since the application currently only runs on Eriksson's P900, the term *cellphone* is referred to as *P900* in the following sections.

Map menu

To start the guide the user is supposed to send a SMS with for instance *GUIDE* to a predefined phone number. This functionality is not implemented. In the current version of the application, the cellphone application starts when the external Java application is started. The phone number to the cellphone is stated in a *.properties* file.

After start-up, the system responds by sending a start page with the map menu as illustrated in Figure 16.9. This page contains links to all available zones. The user can manually navigate to the zone information or automatically get zone information downloaded to the cellphone's browser when he walks in the Cathedral.

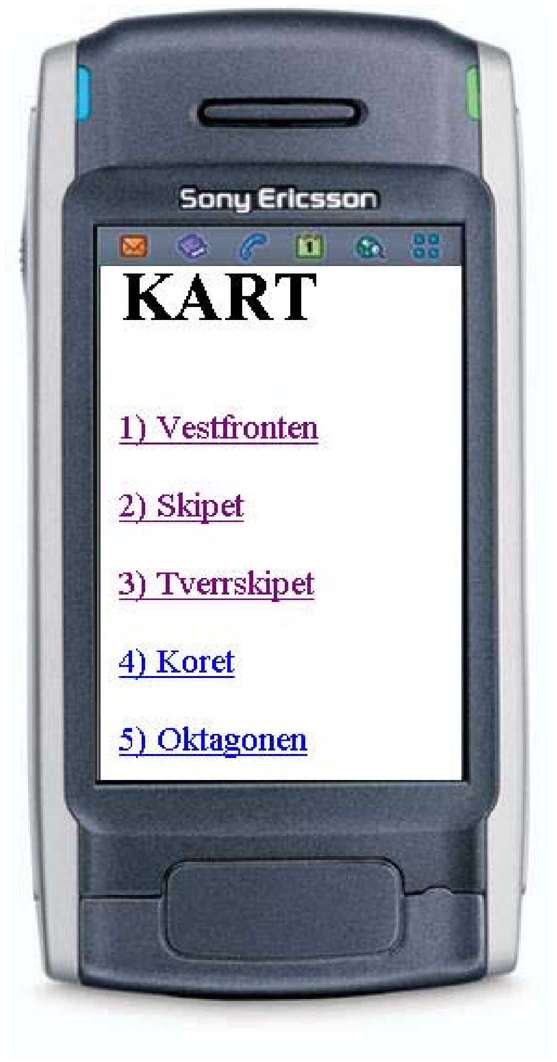


Figure 16.9: Cellphone application - Map

Zone menu

The zone menu shows available objects in a zone. Figure 16.10 illustratest the zone *Koret*. By pressing the objects in the zone menu, the phone will start playing a movie that is streamed as mpeg4 from the web server.

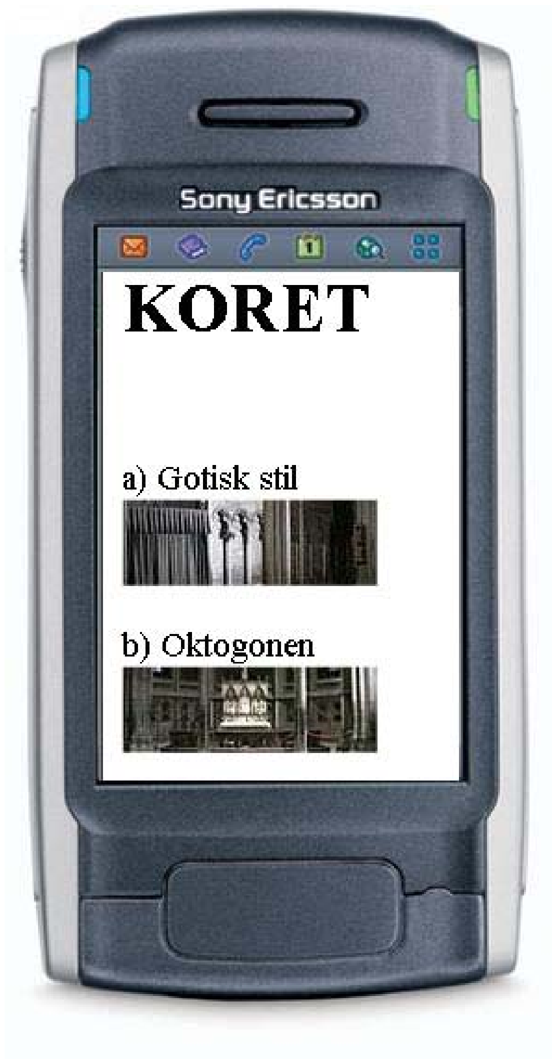


Figure 16.10: Cellphone application - Zone

Object player

In the Nidaros Cathedral there are special objects marked as hotspots. Examples are *døpefonten* and *korsaltert*. When users are getting close to these objects, the phone will automatically play a movie about the object. Figure 16.11 shows an example of a movie that is played in the media player.

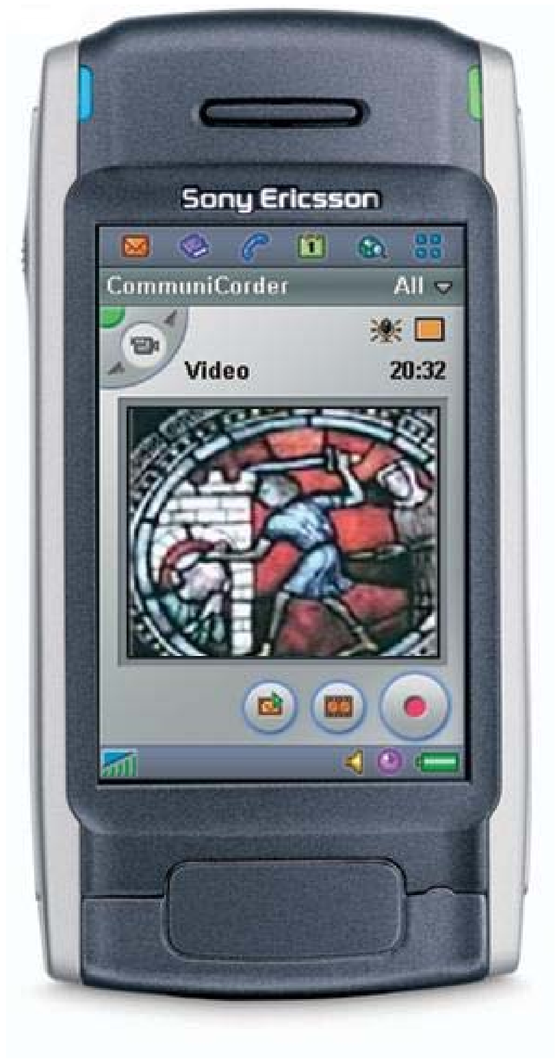


Figure 16.11: Cellphone application - Object

16.3 Summary - Two test applications

The two clients are tested twice in the Nidaros Cathedral. The first at a test day the 28th of April and the second at the demonstration of the system 14th of Mai. The applications are fully operational and can be configured to run by the developers in less then a day.

The client applications demonstrate, by using our framework, some of the possibilities of location-aware tour guides. The PDA application shows that it is possible to use map functionality to track a user and his friends, while the cellphone application demonstrates how future cellphones can be used to automatically send information to the phones adjusted to the user's location.

Chapter 17

Demonstration

On the 14th of May, a presentation of the project was given followed by a demonstration of the system in the Nidaros Cathedral.

17.1 Testing before the demonstration

Before the final demonstration a test was performed in the Nidaros Cathedral. The test's goal was to find out whether it was possible to configure the system to work in the surroundings of the Cathedral. Many people were involved to accomplish the test. Beside us, three persons from Radionor and three persons from Telenor were present. In addition, personnel from the Cathedral were available to help us get access to places to set up the equipment.

The testing took place on the 28th of April. During the testing, four RadioEyes were set up and configured. Afterwards, the necessary measurements were found and the system was configured to fit to the Cathedral.

17.2 Presentation of the project

Before the demonstration in the Cathedral on May 14th, a presentation of the project was given at Gløshaugen. Both the Creator Tool and a simulation of the Runtime System were presented. Approximately 20 people were present, and there were representatives from Ericsson, Radionor, TV Norge, the Nidaros Cathedral, Klipp og Lim Media and NTNU. VG was supposed to be present, but they had to cancel due to strike amongst the journalists. However, we met a journalist from VG a few weeks later and presented the project then. In addition to our presentation, Steinar Brede (our supervisor from Telenor), Alf Inge Wang (our supervisor from NTNU) and Rolv Bræk (from NTNU) also said a few words about the project and the cooperation between the parties involved. Figure 17.1 is a picture from the presentation.

Figure 17.1: Sigurd during the presentation

17.3 Demonstration in the Nidaros Cathedral

After the presentation, we went to the Nidaros Cathedral to show the system in its real surroundings. Both the PDA application and the cellphone application were demonstrated. Figure 17.2 is from the demonstration in the Cathedral.

Figure 17.2: Demonstration in the Cathedral

17.4 Feedback after the demonstration

The people present at the presentation and demonstration were most satisfied with the results of the project. Especially the cellphone application attracted attention. As far as we know, there does not exist any similar applications on cellphones in the world.

A representative from the Cathedral was very exited with the project, and could imagine that a tour guide on cellphones would be a great substitute for the PDA tour guide they use today.

Klipp og Lim Media was also excited with the results, and they are going to continue with related projects.

Ericsson has published information on their local network about our project. They have received a great amount feedback from people who were interested in the project.

17.5 Publications

After the demonstration, we gained publicity about our project. The TV Norge News [1] and NRK Midtnytt have shown a reportage covering the system. These reportages are included on the accompanying CD. In addition, articles are published in Teknisk Ukeblad and Verdens Gang. These can be found in Appendix B and Appendix C.

[1]The TV Norge reportage say we were three students working with the project. However, we are only two.

Part V

Discussion, conclusion and future work

This part sums up the project with a discussion, conclusion and suggestions to future work with the framework.

Chapter 18

Discussion

This chapter presents several aspects regarding the master thesis. Section 18.1 contains a technical discussion, section 18.2 contains personal experiences and section 18.3 contains challenges encountered during the development of the project.

18.1 Technical discussion

This section evaluates the arcitecture, the framework and the implementation.

18.1.1 Evaluation of the architecture

Since the architecture constitutes an important part of the master thesis, the architecture of the framework was carefully developed.

The foundation for the architecture chapter was two articles about software architecture. [Boa00] suggests how the architecture should be documented and [Kru95] explains five different views that can be used in the documentation. By using the IEEE 1471 standard it was easier to make a satisfactory architecture. The standard made sure no important parts were left out. In addition, it provided a well-arranged presentation of the architecture.

18.1.2 About the framework

Our framework for developing location-aware tour guides has several benefits. A system on the server side is prepared and ready; the only information missing is the information about the attraction area. This information can easily be added by using the Creator Tool. In addition, the Creator Tool can be used to make changes to the location information. The framework reduces the expenses of developing location-aware tour guides. As a consequence several attractions may afford such tour guides.

The framework only handles the development of tour guides on the server side. Thus the tour guides have to be developed individually on the client side to fit a specific attraction. This is a time consuming task, which has to be accomplished by a computer engineer. An ideal framework would contain both a system for adding necessary location information on the server side and for generating the applications on the client side.

Our framework contains a service for simulating a user's location. The service will ease the development of a tour guide.

The database that contains the location information is structured as general as possible. However, there are limitations on the information that can be stored. There is no guarantee that the structure will fit every existing attraction.

Because of limited storage room on the clients, a File Server was used in our architecture to store the various presentations of the attraction objects. These files are transferred to the client on request. The transmission of the files made for the Nidaros Cathedral uses on average 5 to 7 seconds, depending of the file sizes. A user has high expectations of the performance and will not have the patience to wait 5 to 7 seconds. Therefore, during the demonstration, described in chapter 17, the most important media files were stored on the client.

Another problem with the client-server solution, is the use of a wireless network. If there are network problems, the guides will not function. There should exist a backup solution on the clients that isn't dependent on the network.

However, despite the problems described above, we belive the system has great potential. When the main problems are solved, many parties can benefit from the system.

The cellphone application is an innovative application that may result in a tremendous evolvement in the development of cellphone applications. Many people have shown interests in it, and the project have gained a great deal of media publicity, both from newspapers and television.

18.1.3 Evaluation of the implementation

Creator Tool

The Creator Tool was developed in Visual Basic (VB) by using Visual Studio 6.0. Visual Studio made it easy to develop the graphical user interface rapidly by using the drag and drop functionality. In addition, the resulting program obtained a typical Microsoft look and feel that many users are familiar with.

The VB language and Visual Studio 6.0 are not the newest products available. A newer version of both VB (Visual Basic .NET) and Visual Studio (Visual Studio .NET) are developed. The VB .NET language provides the traditional ease-of-use of VB development, and allows optional use of new language features like inheritance, method overloading and structured exception handling [Com]. However, the functionality of the VB language was sufficient for the Creator Tool. In addition, one of the authors had knowledge of Visual Studio 6.0. It was therefore decided not to spend any extra time to learn other languages and development tools.

Runtime System

The Runtime System was developed using Java, and runs as a Servlet. The system handles every incoming request as it appears. During the testing several client applications continuously communicated with the Servlet. The Runtime System handled all requests smooth with response time less then a second. Thus, the system has no problems serving a big number of clients.

18.1.4 The use of a Location Server

In the architecture, the framework used a Location Server to receive the user's location. By using a Location Server, the Runtime System gives the responsibility to find the user's location to another part of the system. This divides the functionality in separate modules. The separation makes it easier to perform editing or improvements to the code after its completion date.

The Location Server also makes it easy to add new positioning system to the framework since only code in the Location Server needs to be edited.

18.1.5 The use of XML

XML was used as communication language between the server and the clients. The strengths of XML include its readability and extensibility. In addition, by allowing software to read and write XML documents, the software will be able to communicate with any other software that supports XML.

There are also drawbacks with the use of XML. The XML files can become very large because of the use of tags, and large files are demanding to transmit and parse.

We have experienced that the parsing of the XML data on the client side is a very demanding task. The clients have limited CPU capacity, and the parsing of XML data is a time consuming task. The clients spend more time on parsing the XML data from the server than they use to send a request and receive a response.

18.2 Personal experiences

The following section presents the experience we gained from the cooperation with other parties and trough the demonstration of the system.

18.2.1 The cooperation with so many parties

This project has involved many parties. The project was originally a cooperation between the MOWAHS project, Telenor R&D and Klipp og Lim Media. Due to the demonstration in the Nidaros Cathedral where we used the RadioEye to locate the clients, we also had a close cooperation with Radionor.

The cooperation has been a positive experience for us. There were many people with interests in the final demonstration, and this lead to extra pressure to develop a satisfactory framework. Telenor R&D, Klipp og Lim Media, Radionor and NTNU would all benefit from a successful demonstration. All the pressure lead to us working harder, which further resulted in a very successful demonstration.

18.2.2 The demonstration in the Nidaros Cathedral

The demonstration in Nidaros Cathedral required a great amount preparation. Two weeks of the project period were dedicated to this demonstration. In this period the code was tested and improvements were carried out, and we cooperated with Radionor to configure the RadioEyes as optimal as possible.

The lesson learned from this experience is that it takes a great amount of preparation to perform a demonstration of a product.

18.3 Challenges

The main challenge of this thesis was the development of the architecture of the framework. The architecture was of great importance to the successfulness of the system, and had to be carefully designed. When the implementation started, it was difficult to go back to the architecture and do any changes.

There were several challenges with a project that has so many parties involved. To find time to have a meeting where all parties could participate wasn't always easy. Hence, some of the meetings were accomplished without all the parties present.

The demonstration in the Nidaros Cathedral gave us technical challenges. Necessary equipment had to be set up and the framework had to be configured. Since we didn't have any measures of the Cathedral, we had to measure the necessary values ourselves. Before the demonstration, it had to be planned who was responsible for the various parts, and who was going to present the various parts.

Chapter 19

Conclusion

Today, several tourist attractions use handheld devices that act as location-aware tour guides to give the customers an improved experience and better knowledge of the attraction. However, there exist few frameworks supporting the development of such systems.

The main goal of this master thesis was to develop a *Generic framework for development of location-aware applications*. Parts of the framework were going to be implemented, and a demonstrator was going to be developed to prove the concept. We conclude that the goal is achieved.

Our framework consists of three main tools; a *Creator Tool*, a *Statistical Tool* and a *Runtime System*. The *Creator Tool* is used to create and configure new tour guides. The *Statistical Tool* is used by the staff of the attraction. It generates statistics based on information stored in a log. The *Runtime System* is providing the mobile devices with information adjusted to their location during a guided tour.

Selected parts of the Creator Tool and the Runtime System have been implemented, while the Statistical Tool is not implemented. We have developed two client applications in order to demonstrate the use of the framework, one for PDAs and the other for cellphones. Both applications are electronically location-aware tour guides tailored for the Nidaros Cathedral.

The applications have been tested at the Nidaros Cathedral in two occasions. The cellphone application is the first of its kind, and the system gained a lot of media interest. It has been presented on television and written about in papers and magazines.

Throughout the work of this master thesis we have shown the possibilities of using location-aware applications and the importance of having a framework to support the development of such systems. A framework will reduce the development cost and the development time of new tour guides. The project has opened doors for projects in the area of location-aware tour guides. Klipp og Lim Media will try to make a commercial product available based and the results of this master thesis..

Chapter 20

Future work

This project resulted in a prototype of a framework for developing location-aware applications. The first section in this chapter debates the possibilities for future work with this framework, while the last section presents Klipp og Lim Media, NTNU and Telenor R&D's future plans based on the master thesis.

20.1 Future work with the prototype

The prototype is not a complete system ready to be commercialized. The following sections present some future work that can be done and provides examples of other areas where the framework may be used

20.1.1 The requirements

Not all functional requirements in chapter 8 are implemented. In a final version all the requirements should be fulfilled and thoroughly tested. Amongst other things, the Statistical Tool needs to be implemented. In addition, it may be desirable to extend the framework's functionality with more services.

An example of a new service for the employees in tourist attraction can be to make it possible to broadcast messages to the visitors in the attraction. Interesting messages to broadcast can be ”We are closing in 10 minutes” or messages about special events taking place.

In addition, the functionality could be extended to add information about how to locate the nearest toilet, food store or similar, based on users location.

20.1.2 Runtime System

The database is using a data type called *Geometry*. There are supposed to be some functions available to this data type, but these functions are not yet implemented in the MySQL database. Therefore, Java is used in the prototype to do calculations regarding positioning topics. In a future version, the database itself can performe these calculations.

20.1.3 Creator Tool

The Creator Tool only deals with aspects regarding the server side of the framework. As mentioned in chapter 18, an ideal implementation of the Creator Tool would constitute of a system for both handling the location information on the server side, and for generating the tour guide applications on the client side. By adding the functionality for generating guide applications on the client side, the framework would constitute a total system for development of location-aware applications.

In the current version of Creator Tool, the users manually have to measure and calculate coordinates for each object in the tour guide. This is a very time consuming task and to reduce the effort this should be automized. A way to do this is to automatically gather position data through a mobile device when moving around on the tourist attraction.

20.1.4 The Location Server

During the development and testing of the prototype, we did not use the Location Server described in the architecture. Instead, we used the RadioEye to retrieve the user's location.

The Location Server is supposed to consist of several positioning technologies. When the Runtime System sends a request for a user's location, the Location Server examines which positioning technology provides the most accurate results and returns this position. We have not been able to implement and test this functionality. However, as described in the architecture, it would be an advantage to use a Location Server in the framework.

20.1.5 Other tour attractions

This project has developed client applications for one attraction, namely the Nidaros Cathedral. The goal of the framework was to generate tour guides for several attractions, but this has not been tested. The architecture of both the framework and the database is made to be as general as possible to fit most attractions. Therefore, there are no problems in developing a tour guide for another attraction.

20.1.6 Range of use

In addition to provide location information to tourists as in the Nidaros Cathedral demonstration, the framework can provide location information to mobile workers as well. Such a system can typically be used to educate new employees and guide them through their tasks based on their location. The framework can also provide mobile workers information about safety boundaries and danger zones. In some environments, the mobile worker should be warned if he walks outside a safe area and must take precautions.

In general, the framework can enhance user interfaces for mobile devices by displaying only the relevant information at a respective location. A common problem with mobile applications running on mobile devices is the small screens. By utilizing the location, irrelevant information can be left out for a better usage of the screen. Typical users of the framework could be construction workers, repairmen on shipyards and platforms, electricians, plumbers, home care nurses, physicians and emergency workers.

20.2 Future work at NTNU, Telenor R&D and Klipp og Lim Media

The participants involved found the project to be a success, and want to continue to work within the topic of this master thesis.

NTNU and Telenor R&D have already planed a new cooperation next semester with new students. There will be at least one project with a topic related to this master thesis.

Klipp og Lim Media is so pleased with the prototype developed that they have decided to start a project inspired from this master thesis.

Bibliography

[Boa00] IEEE-SA Standards Board. Ieee recommended practice for architectural description of software-intensive systems. 2000.

[Bon04] Kevin Bonsor. How location tracking will work, February 2004. URL: www.people.howstuffworks.com/locationtracking1.htm.

[CMEE04] The Physics Classroom and Inc. Mathsoft Engineering & Education. Ultrasound, February 2004. URL: http://www.physicsclassroom.com/Class/sound/U11L2a.html.

[CN] Reidar Conradi and Mads Nygård. Mowahs - mobile work across heterogeneous systems. Available from http://www.idi.ntnu.no/grupper/su/mowahs.html.

[CNS] George Coulouris, Hani Naguib, and Kam Samugalingam. Flame: An open framework for location-aware systems. Available from http://www-lce.eng.cam.ac.uk/qosdream/Publications/flame.pdf.

[Com] GotDotNet Community. About visual basic .net. Available from http://www.gotdotnet.com/team/vb/.

[Com04] Radionor Communications. Cordis radioeye, March 2004. URL: http://www.radionor.no/.

[CP03] F Garzotto; T Cinotti and M Pigozzi. Designing multi-channel web frameworks for cultural tourism applications: the muse case study. *PAPERS Museum and the Web 2003*, 2003.

[Dan04] Peter H. Dana. Coordinate systems overview, April 2004. URL: http://www.colorado.edu/geography/gcraft/notes/coordsys/coordsys.html.

[DCMC01] Sastry Duri, Alan Cole, Jonathan Munson, and Jim Christensen. An approach to providing a seamless end-user experience for location-aware applications. 2001.

[DCME01] Nigel Davids, Keith Cheverset, Keith Micthell, and Alon Efrat. Using and determining location in a context-sensitive tour guide. *IEEE*, August 2001.

[Eka04] Ekahau. Ekahau, March 2004. URL: http://www.ekahau.com/.

[Eri01] Ericson. Mobile positioning protocol. *Geodetic datum and Coordinate systems*, 2001. version 4.0.

[Fra04] Curt Franklin. How bluetooth works, February 2004. URL: http://www.people.howstuffworks.com.

[Gar04] Garmin. Hva er gps, January 2004. URL: http://home.no.net/perfrode/Kart/hva_er_gps_paa_norsk.htm.

[GD98] Green and DiCaterino. A survey of system development process models. *Center for Technology in Government*, pages 1–13, February 1998.

[Geo04a] Geodata. Hvor.no, February 2004. URL: http://www.hvor.no.

[Geo04b] Geodesi. Rikets kartprojektionssystem, April 2004. URL: http://www.lm.se/geodesi/refsys/rt/projektioner_i_rt.htm.

[Gil04] Tom Gilb. *Competitive Engineering*. Only available on net, 2004. URL: http://www.gilb.com.

[GS03] Sigurd Gimre and Hege Servold. Mowahs - location-based guiding. Master's thesis, Norwegian University of Science and Technology, Desember 2003. URL: http://www.idi.ntnu.no/grupper/su/fordypningsprosjekt-2003/fordypning2003-Sigurd-Gimre-og-Hege-Servold.pdf.

[HT03] Ole Hestnes and Runas Mehl Teigen. Anvendelse av metodikken contextual design for utvikling av lokasjonsbaserte tjenester ved grimstad museum. Master's thesis, Agder University college, June 2003. URL: http://siving.hia.no/ikt03/ikt6400/g23.

[Ita04] Itavisen. 108 mbps trådløst nett, March 2004. URL: http://www.itavisen.no/art/1303051.html.

[KP97] Gregory Abowd; Christopher Atkeson; Jason Hong; Sue Long; Rob Kooper and Mike Pinkerton. Cyberguide: A mobile context-aware guide. *Wireless Networks*, (3), 1997.

[Kru95] Philippe Kruchten. The 4+1 view model of software architecture. november 1995.

[Lab04] AT&T LaboratoriesCambridge. The bat ultrasonic location system, February 2004. URL: http://www.uk.research.att.com/bat/.

[LB03] Paul Clements ad Rich Kazman Len Bass. *Software Architecture in Practice - Second Edition*. Addison-Wesley, 2003.

[Lip04] Lipus. Ultrasound, February 2004. URL: http://www.lipus.info/us/ultrasound

[Mol03] Eivind Molstad. Mowahs - location-based applications for pda, mobile terminals. Master's thesis, Norwegian University of Science and Technology, June 2003. URL: http://www.idi.ntnu.no/grupper/su/su-diploma-2003/Molstad-Location-basedAccommodationService.pdf.

[MT02] E. Bonek M Taferner. *Wireless Internett Access over GSM and UTMS*. Springer Verlag, April 2002. ISBN 3-540-42551-9.

[MyS] MySQL. Mysql - the world's most popular open source database. Available from http://www.mysql.com/.

[Net04] Netscape. Buddy, February 2004. URL: http://www.netcom.no/ default.asp?Identificator=1.1.2.1&DocumentId=174.

[Som01] Ian Sommerville. *Software Engineering*. Addison-Wesley, 2001. ISBN: 0-201-39815-X.

[tec04] Sonitor technologies. Sonitor tracing systems, February 2004. URL: http://www.sonitor.com/.

[Tel] Telenor. Arts - arena for research on advanced telecom services. Available from http://www.pats.no/projects/ARTS/arts.html.

[thg04] tom's hardware guide. Die 108-mbit/s-pc-card wg511t, March 2004. URL: hhttp://www.de.tomshardware.com/network/20031024/netgear-06.html.

[VS00] Steffen Volz and Monika Sester. Nexus - distributed management concepts for location aware applications, 2000. Available from http://www.ifp.uni-stuttgart.de/publications/2000/Volz_Ascona_00.pdf.

[Wan] Alf Inge Wang. Mobile work across heterogeneous systems. Available from http://www.mowahs.com/.

[WH92] R. Want and A. Hopper. Active badges and personal interactive computing objects. *IEEE Transactions on Consumer Electronics*, February 1992.

[WHFG92] R. Want, A. Hopper, V. Falcão, and J Gibbons. The active badge location system. *Olivetti Research Ltd. (ORL) Cambridge, England*, January 1992.

[WJCK02] R. Woodings, D. Joos, T. Clifton, and C.D. Knutson. Rapid heterogeneous ad hoc connection establishment: Accelerating bluetooth inquiry using irda. *Proceedings of the Third Annual IEEE Wireless Communications and Networking Conference (WCNC '02), Orlando*, March 2002.

Part VI

Appendix

Appendix A

Press release

This appendix contains the information handed to the participators of the press conference the 14th of Mai. The text is quoted as released and therefore written in Norwegian.

Translations:
The term *turistguide system* refer to *Runtime System*

The term *konfigurasjons verktøy* refer to *Creator Tool*

Stedsfølsom turistguide” på mobiltelefon og PDA

4.Mai 2004

ARTS Prosjektet

Arena for Research on advanced Telecom Services
NTNU, Telenor FoU, Abelia, Ericsson (2003-2006)

ARTS er et samarbeidsprosjektet mellom NTNU, Telenor FoU, Ericsson og Abelia Innovasjon med støtte fra Norges Forskningsråd. Prosjektet adresserer de teknologiske utfordringer rundt nye avanserte teletjenester. Praktisk utprøving av disse tjeneste bli utført i PATS labben (www.pats.no) og det er etablert et innovasjonssenter for tredjeparts tjenesteutviklere.

Definisjonen av prosjektet har vært gjennomført i samråd med Telenor Mobil. De har allerede i dag stor pågang fra tredjeparts tjenesteutviklere og undervisningsmiljøer om tilgang til både plattformer og nettressurser, og støtter opp om dette labbmiljøet for eksperimentering. Laboratoriet vil i et tidlig stadium reise praktiske problemstillinger rundt tredjeparts tjenesteutvikling. Det inkluderer blant annet administrasjon, sikkerhet, pålitelighet og forretningsmodeller.

Målsettinger

Prosjektets målsettinger er delt i to kategorier.

- Forskning, utvikling og undervisning
- Innovasjonssentret.

Forskning, utvikling og undervisning

Prosjektet skal adressere metoder og verktøy for hurtig tjenesteutvikling og tjenesteeksekveringsplattformer. I tillegg skal prosjektet henvende seg mot forretningsmessige aspekt som forretningsmodeller og SLA for tredjeparts tilgang mot telenettet. Prosjektet skal finansiere 2 Dr.ing. stipendiater og vil knytte til seg minst 10 NTNU studenter i form av høst- og diplomoppgaver.

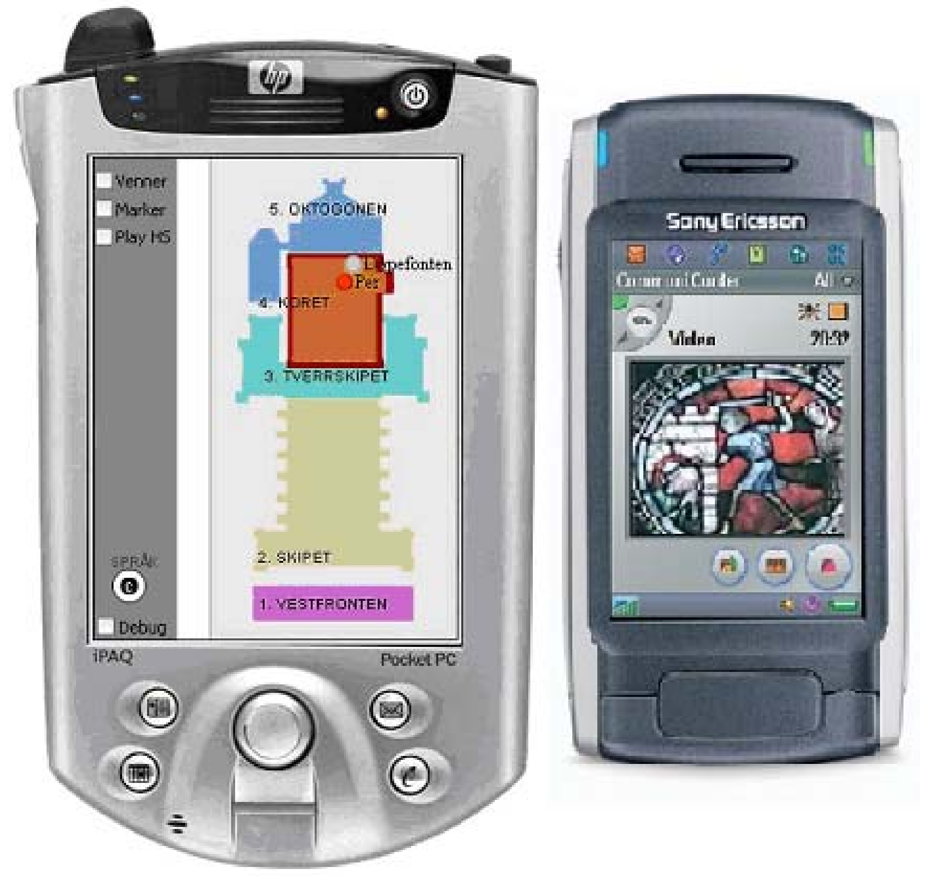
Innovasjonssenter

Som et ledd i prosjektet vil PATS lab bli benyttet som et innovasjonssenter som skal knytte til seg et antall tredjeparts firmaer for prototyping, eksperimentering og ”feasability” studier. Innovasjonssentret skal tilby ”bundlede” tjenestekomponenter mot ulike bransjer, som transport, helse og undervisning. Prosjektet vil holde workshops for tredjepart tjenestutviklere og kunne tilby supporttjenester for tjenesteutvikling.

Kontaktpersoner:

Telenor FoU (prosjektleder)	Steinar Brede, steinar.brede@telenor.com
Ericsson Research	Geir Melby, geir.melby@ericsson.com
NTNU - ITEM:	Rolv Bræk, rolv.braek@item.ntnu.no
Abelia	Geir Amsjø, geir.amsjo@abelia.no
Web-side:	http://www.pats.no/

Stedsfølsom turistguide i Nidaros-domen



Demonstratoren er utviklet av Hege Servold og Sigurd Gimre (NTNU) i samarbeid med Telenor FoU, Klipp og Lim Media og Radionor.

Prosjektet er Sigurd og Hege sin hovedoppgave i sivilingeniørutdannelsen som de avslutter denne våren. Oppgaven har gått ut på å lage et rammeverk for utvikling av lokasjonsbaserte guider. Guiden i Nidarosdomen er et eksempel på hva rammeverket kan brukes til.

Prosjektet har tatt utgangspunk i den eksisterende turistguide applikasjonen som er laget av Klipp og Lim Media AS. Ved hjelp av WLAN posisjoneringsteknologi fra Radionor er denne applikasjonen videreutviklet til å være ”stedfølsom”.

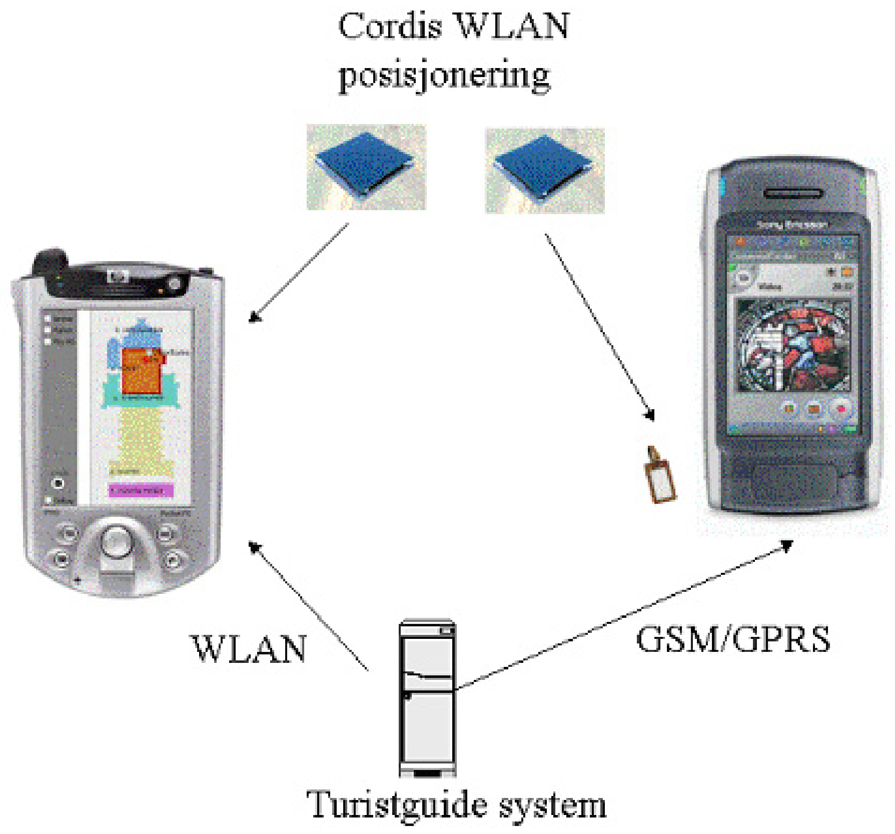
Rammeverket består av tre deler, hvorav kun to er implementert. Disse er:

- Et turistguide system

- Et konfigurasjons verktøy
- Et statistikk verktøy (som ikke er implementert)

Turistguide systemet ligger på en server, og det er dette systemet som brukes når man går rundt i kirken. På denne serveren ligger all informasjon om severdighetenes lokasjon lagret, og i tillegg holder serveren oversikt over hvor brukerne er lokalisert. Informasjonen overføres automatisk til de håndholdte enhetene etter hvert som brukeren beveger seg. På denne måten får brukeren informasjon tilpasset sin egen lokasjon og slipper å søke gjennom stor mengder data for å finne den informasjonen de står i nærheten av.

Turistguide systemet støtter bruken av ulike typer håndholdte enheter, og blir demonstrert på både PDA med WLAN aksess og GSM/GPRS mobiltelefon.

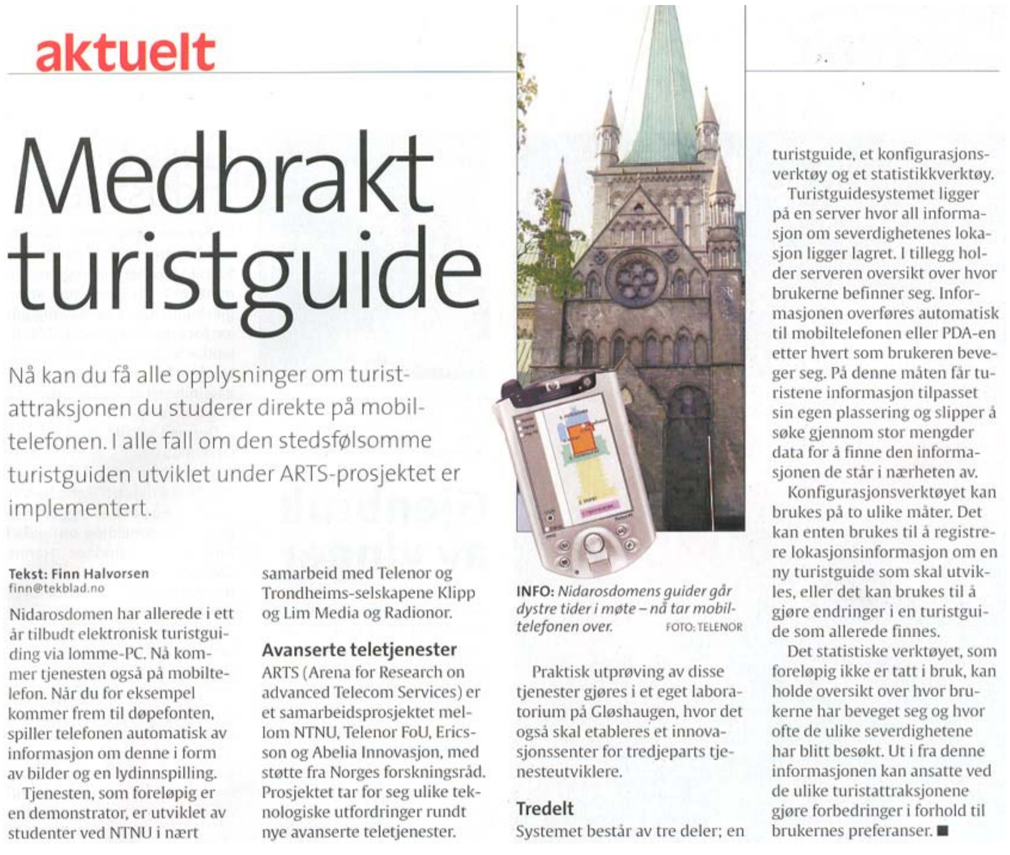


Konfigurasjonsverktøyet kan brukes på to ulike måter. Det kan enten brukes til å registrere lokasjons informasjon om en ny turist guide som skal utvikles, eller det kan brukes til å gjøre endringer i en turist guide som allerede finnes. For eksempel kan det i mange tilfeller være aktuelt å flytte på en severdighet, for eksempel et bilde eller liknende. Med et slikt verktøy, kan man raskt og enkelt utvikle og konfigurere lokasjonsbaserte turistguider. Brukerne av konfigurasjonsverktøyet vil være de som administrerere en severdighet.

Det statistiske verktøyet brukes til å holde oversikt over hvor brukerne har beveget seg og hvor ofte de ulike severdighetene har blitt besøkt. Ut i fra denne informasjonen kan ansatte ved de ulike turist attraksjonene gjøre forbedringer til både attraksjonen og guiden i forhold til brukernes preferanser.

Appendix B

Article from Teknisk Ukeblad



aktuelt

Medbrakt turistguide

Nå kan du få alle opplysninger om turistattraksjonen du studerer direkte på mobiltelefonen. I alle fall om den stedsfølsomme turistguiden utviklet under ARTS-prosjektet er implementert.

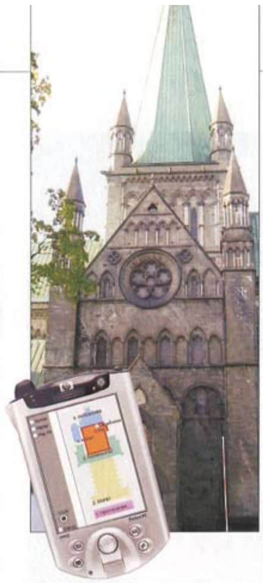
Tekst: Finn Halvorsen
finn@tekblad.no

Nidarosdomen har allerede i ett år tilbudt elektronisk turistguiding via lomme-PC. Nå kommer tjenesten også på mobiltelefon. Når du for eksempel kommer frem til døpefonten, spiller telefonen automatisk av informasjon om denne i form av bilder og en lydinnspilling.

Tjenesten, som foreløpig er en demonstrator, er utviklet av studenter ved NTNU i nært samarbeid med Telenor og Trondheims-selskapene Klipp og Lim Media og Radionor.

Avanserte teletjenester

ARTS (Arena for Research on advanced Telecom Services) er et samarbeidsprosjektet mellom NTNU, Telenor FoU, Ericsson og Abelia Innovasjon, med støtte fra Norges forskningsråd. Prosjektet tar for seg ulike teknologiske utfordringer rundt nye avanserte teletjenester.



INFO: *Nidarosdomens guider går dystre tider i møte – nå tar mobiltelefonen over.* FOTO: TELENOR

Praktisk utprøving av disse tjenester gjøres i et eget laboratorium på Gløshaugen, hvor det også skal etableres et innovasjonssenter for tredjeparts tjenesteutviklere.

Tredelt

Systemet består av tre deler; en turistguide, et konfigurasjonsverktøy og et statistikkverktøy.

Turistguidesystemet ligger på en server hvor all informasjon om severdighetenes lokasjon ligger lagret. I tillegg holder serveren oversikt over hvor brukerne befinner seg. Informasjonen overføres automatisk til mobiltelefonen eller PDA-en etter hvert som brukeren beveger seg. På denne måten får turistene informasjon tilpasset sin egen plassering og slipper å søke gjennom stor mengder data for å finne den informasjonen de står i nærheten av.

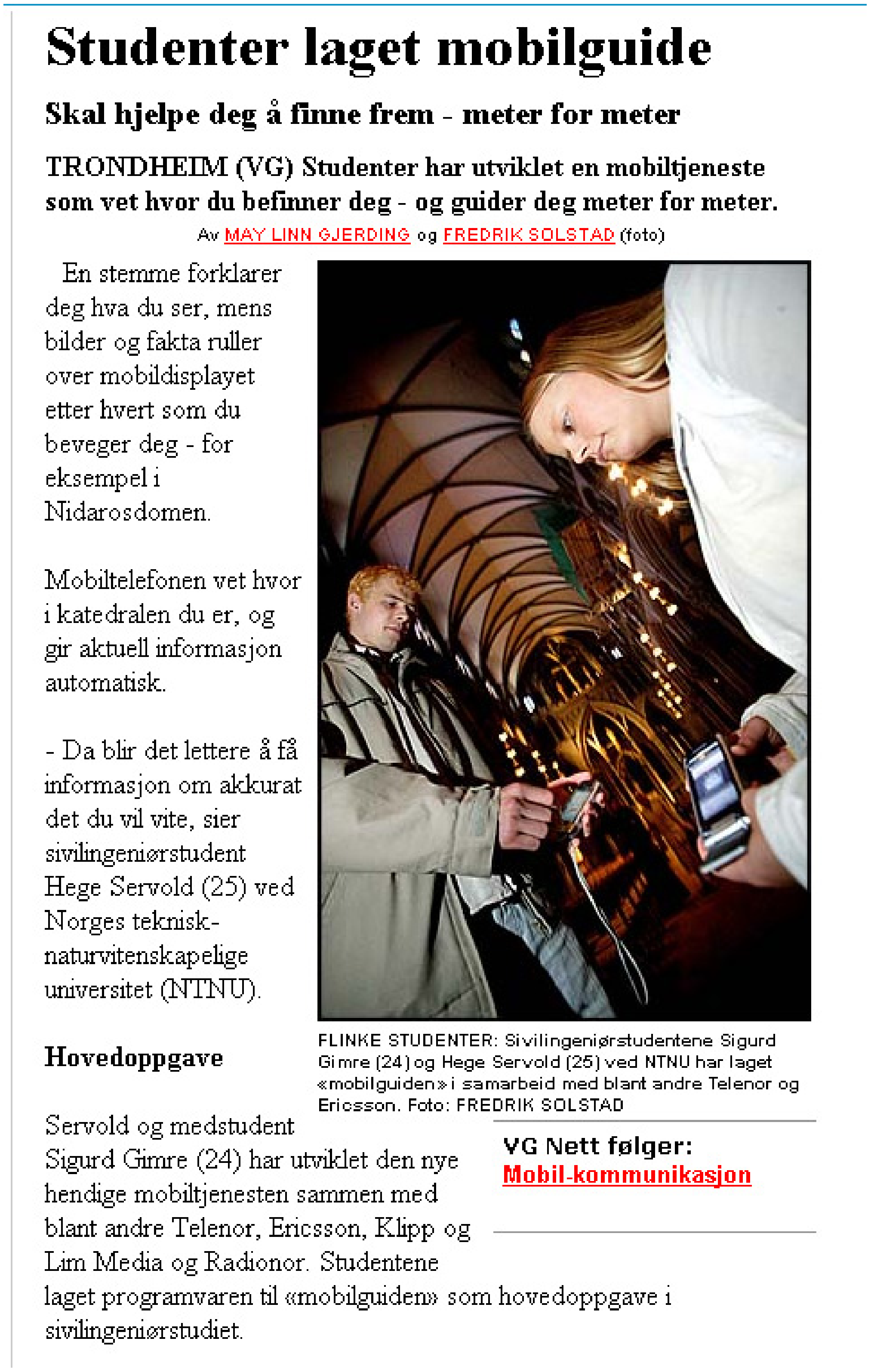
Konfigurasjonsverktøyet kan brukes på to ulike måter. Det kan enten brukes til å registrere lokasjonsinformasjon om en ny turistguide som skal utvikles, eller det kan brukes til å gjøre endringer i en turistguide som allerede finnes.

Det statistiske verktøyet, som foreløpig ikke er tatt i bruk, kan holde oversikt over hvor brukerne har beveget seg og hvor ofte de ulike severdighetene har blitt besøkt. Ut i fra denne informasjonen kan ansatte ved de ulike turistattraksjonene gjøre forbedringer i forhold til brukernes preferanser. ■

Figure B.1: Article from Teknisk Ukeblad

Appendix C

Article from Verdens Gang



Studenter laget mobilguide

Skal hjelpe deg å finne frem - meter for meter

TRONDHEIM (VG) Studenter har utviklet en mobiltjeneste som vet hvor du befinner deg - og guider deg meter for meter.

Av MAY LINN GJERDING og FREDRIK SOLSTAD (foto)

En stemme forklarer deg hva du ser, mens bilder og fakta ruller over mobildisplayet etter hvert som du beveger deg - for eksempel i Nidarosdomen.

Mobiltelefonen vet hvor i katedralen du er, og gir aktuell informasjon automatisk.

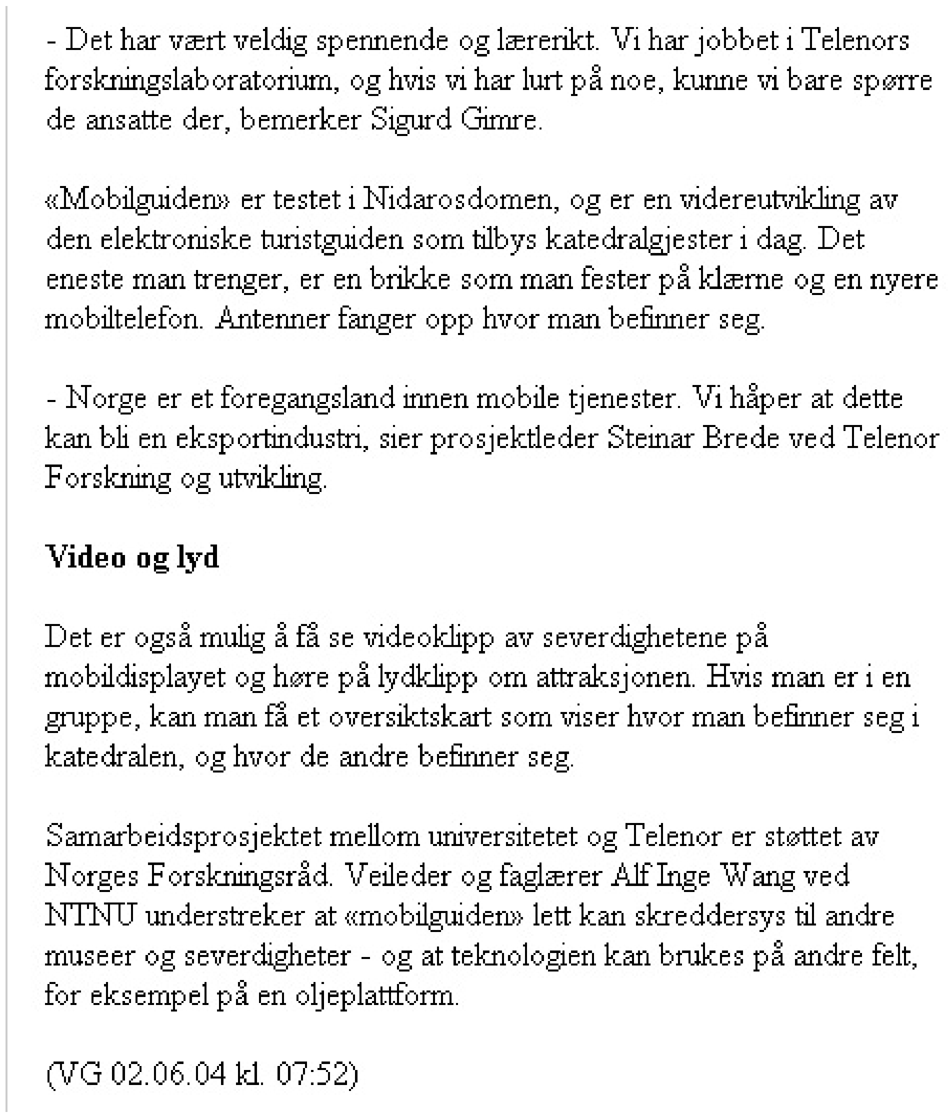
- Da blir det lettere å få informasjon om akkurat det du vil vite, sier sivilingeniørstudent Hege Servold (25) ved Norges teknisk-naturvitenskapelige universitet (NTNU).

FLINKE STUDENTER: Sivilingeniørstudentene Sigurd Gimre (24) og Hege Servold (25) ved NTNU har laget «mobilguiden» i samarbeid med blant andre Telenor og Ericsson. Foto: FREDRIK SOLSTAD

Hovedoppgave

Servold og medstudent Sigurd Gimre (24) har utviklet den nye hendige mobiltjenesten sammen med blant andre Telenor, Ericsson, Klipp og Lim Media og Radionor. Studentene laget programvaren til «mobilguiden» som hovedoppgave i sivilingeniørstudiet.

VG Nett følger:
Mobil-kommunikasjon



- Det har vært veldig spennende og lærerikt. Vi har jobbet i Telenors forskningslaboratorium, og hvis vi har lurt på noe, kunne vi bare spørre de ansatte der, bemerker Sigurd Gimre.

«Mobilguiden» er testet i Nidarosdomen, og er en videreutvikling av den elektroniske turistguiden som tilbys katedralgjester i dag. Det eneste man trenger, er en brikke som man fester på klærne og en nyere mobiltelefon. Antenner fanger opp hvor man befinner seg.

- Norge er et foregangsland innen mobile tjenester. Vi håper at dette kan bli en eksportindustri, sier prosjektleder Steinar Brede ved Telenor Forskning og utvikling.

Video og lyd

Det er også mulig å få se videoklipp av severdighetene på mobildisplayet og høre på lydklipp om attraksjonen. Hvis man er i en gruppe, kan man få et oversiktskart som viser hvor man befinner seg i katedralen, og hvor de andre befinner seg.

Samarbeidsprosjektet mellom universitetet og Telenor er støttet av Norges Forskningsråd. Veileder og faglærer Alf Inge Wang ved NTNU understreker at «mobilguiden» lett kan skreddersys til andre museer og severdigheter - og at teknologien kan brukes på andre felt, for eksempel på en oljeplattform.

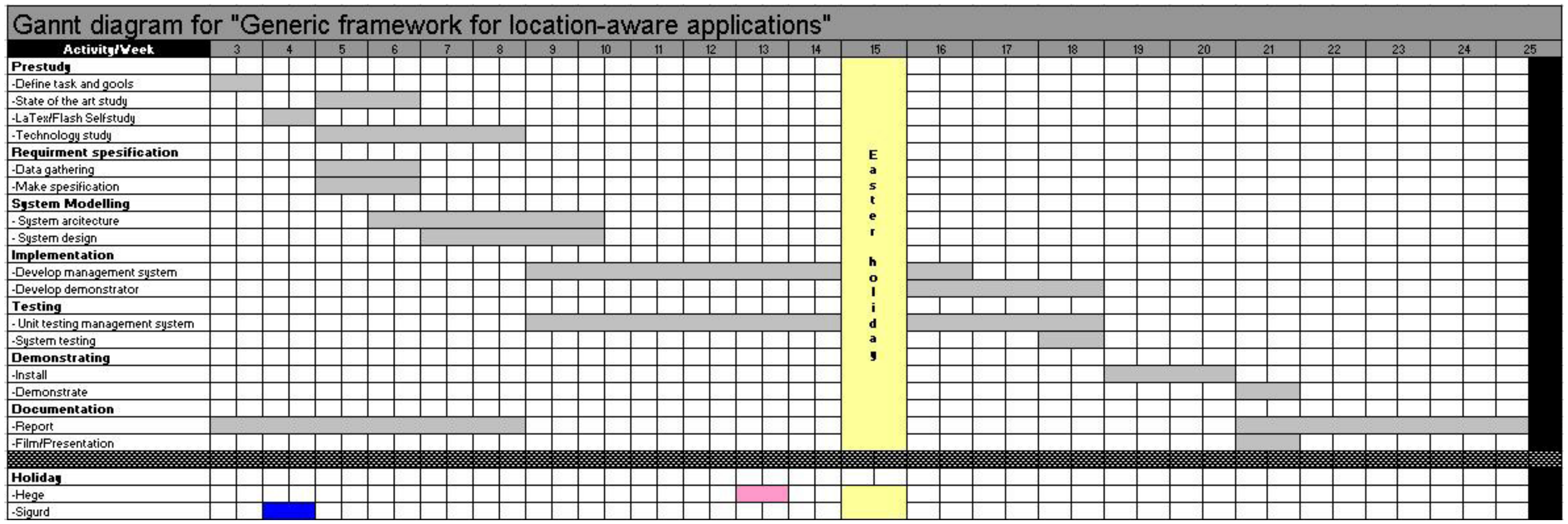
(VG 02.06.04 kl. 07:52)

Figure C.1: Article from Verdens Gang

Appendix D

Gannt Diagram

This Appendix presents the Gannt diagram used during the project execution.



Gannt diagram for "Generic framework for location-aware applications"

Activity/Week	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Prestudy													Easter holiday										
-Define task and gools	▨																						
-State of the art study			▨	▨																			
-LaTex/Flash Selfstudy		▨																					
-Technology study			▨	▨	▨	▨																	
Requirment spesification																							
-Data gathering			▨	▨																			
-Make spesification			▨	▨																			
System Modelling																							
- System arcitecture				▨	▨	▨	▨	▨															
- System design					▨	▨	▨	▨															
Implementation																							
-Develop management system							▨	▨	▨	▨	▨	▨		▨									
-Develop demonstrator														▨	▨	▨							
Testing																							
- Unit testing management system							▨	▨	▨	▨	▨	▨		▨	▨	▨							
-System testing																▨							
Demonstrating																							
-Install																	▨	▨					
-Demonstrate																			▨				
Documentation																							
-Report	▨	▨	▨	▨	▨	▨													▨	▨	▨	▨	▨
-Film/Presentation																			▨				
Holiday																							
-Hege											▨		▨										
-Sigurd		▨											▨										

Figure D.1: Gannt Diagram

Appendix E

DTD description

This Appendix presents the DTD used to validate the XML communication between the *Runtime System* and the *Client Applications*.

locationRequest DTD

```
<?xml version="1.0" encoding="UTF-8"?>

<!--This DTD describe the protocol for the locationRequest
sent to the Runtime System by the Client Applications that
use the "Generic framework for development of location
-aware tour guides".The protocol should be used to
validate that the request is sent in correct format.
It gives a easy structural explanation of the
XML request. -->

<!ELEMENT locationRequest (mac, get+, mapping*)>
<!ELEMENT mac (#PCDATA)>
<!ELEMENT get (#PCDATA | name | startX | startY
| stepX | stepY | stopX | stopY)*>
<!ELEMENT mapping (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT startX (#PCDATA)>
<!ELEMENT startY (#PCDATA)>
<!ELEMENT stepX (#PCDATA)>
<!ELEMENT stepY (#PCDATA)>
<!ELEMENT stopX (#PCDATA)>
<!ELEMENT stopY (#PCDATA)>
```

locationResponse DTD

```
<?xml version="1.0" encoding="UTF-8"?>

<!--This DTD describe the protocol for the locationResponse
returned from the Runtime System to Client Applicatoins using
the Generic framework for development of location-aware tour guides.
It is used to validate correct response from the RS.
 -->

 <!-- The DTD validates the response to every possible
  locationRequest-->

<!ELEMENT locationResponse(position,dynamicInfo,
friendPosition,tracks,message,error)>

<!ELEMENT position (user, map, zone, hotSpotObject)>
<!ELEMENT user (mac?, mapping?, nick?, x, y, map?)>
<!ELEMENT map (name, number, floor, zone*)>
<!ELEMENT zone (name, number, floor?, located?, object*)>
<!ELEMENT hotSpotObject (name, number, x, y)>
<!ELEMENT mac (#PCDATA)>
<!ELEMENT mapping (#PCDATA)>
<!ELEMENT nick (#PCDATA)>
<!ELEMENT x (#PCDATA)>
<!ATTLIST x id CDATA #IMPLIED>
<!ELEMENT y (#PCDATA)>
<!ATTLIST y id CDATA #IMPLIED >
<!ELEMENT name (#PCDATA)>
<!ELEMENT number (#PCDATA)>
<!ELEMENT floor (#PCDATA)>

<!ELEMENT dynamicInfo (map+)>
<!ELEMENT located (#PCDATA)>
<!ELEMENT object (name, number, x, y, hs, url)>
<!ELEMENT url (#PCDATA)>
<!ELEMENT hs (#PCDATA)>

<!ELEMENT friendPosition (user+)>

<!ELEMENT tracks (track)>
<!ELEMENT track (name, number, coord)>
<!ELEMENT coord (x | y)+>

<!ELEMENT message (note+)>
<!ELEMENT note (to, from, header, body)>
<!ATTLIST note id CDATA #REQUIRED>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
```

```
<!ELEMENT header (#PCDATA)>
<!ELEMENT body (#PCDATA)>

<!ELEMENT error (type, text)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT text (#PCDATA)>
```

Appendix F

XML examples

This Appendix provides several examples on different location requests form client applications with their belonging location responses from the Runtime System.

<get>position</get>

```
<locationRequest>
     <mac>00:0B:FD:C6:5C:AB</mac>
     <get>position</get>
     <mapping>PDA</mapping>
</locationRequest>
```

```
<locationResponse>
     <position>
         <user>
              <mac>00:0B:FD:C6:5C:AB</mac>
              <mapping>PDA</mapping>
              <x>100</x>
              <y>130</y>
         </user>
         <map>
              <name>Første venstre</name>
              <number>1</number>
              <floor>1</floor>
         </map>
         <zone>
              <name>Inngang</name>
              <number>3</number>
         </zone>
         <hotSpotObject>
              <name>Statue 1</name>
              <number>1</number>
              <x>100</x>
              <y>120</y>
         </hotSpotObject>
      </position>
</locationResponse>


<locationResponse>
     <position>
          <user>
               <mac>00:0B:FD:C6:5C:AB</mac>
               <mapping>PDA</mapping>
               <x>100</x>
               <y>130</y>
          </user>
          <map>noinfo</map>
          <zone>noinfo</zone>
          <hotSpotObject>noinfo</hotSpotObject>
     </position>
</locationResponse>
```

<get>simulatedPosition</get>

```
<locationRequest>
        <mac>BF:HJ:JK:KJ</mac>
        <get>simulatedPosition
               <startX>25</startX>
               <startY>50</startY>
               <stopX>50</stopX>
               <stopY>110</stopY>
               <stepX>1</stepX>
               <stepY>1</stepY>
        </get>
</locationRequest>
```

```
<locationResponse>
     <position>
         <user>
              <mac>00:0B:FD:C6:5C:AB</mac>
              <mapping>PDA</mapping>
              <x>100</x>
              <y>130</y>
         </user>
         <map>
              <name>Første venstre</name>
              <number>1</number>
              <floor>1</floor>
         </map>
         <zone>
              <name>Inngang</name>
              <number>3</number>
         </zone>
         <hotSpotObject>
              <name>Statue 1</name>
              <number>1</number>
              <x>100</x>
              <y>120</y>
         </hotSpotObject>
      </position>
</locationResponse>
```

<get>friendPosition</get>

```
<locationRequest>
     <mac>00:0B:FD:C6:5C:AB</mac>
     <get>friendPosition</get>
     <mapping>PDA</mapping>
</locationRequest>
```

```
<locationResponse>
     <friendPosition>
          <user>
               <nick>truls</nick>
               <x>100</x>
               <y>130</y>
               <map>
                    <name>Map 1</name>
                    <number>1</number>
                    <floor>1</floor>
               </map>
          </user>
          <user>
               <nick>fredrik</nick>
               <x>-1</x>
               <y>-1</y>
               <map>
                    <name>noinfo 1</name>
                    <number>-1</number>
                    <floor>1</floor>
               </map>
          </user>
     </friendPosition>
</locationResponse>
```

<get>dynamicInfo</get>

```
<locationRequest>
     <mac>00:0B:FD:C6:5C:AB</mac>
     <get>dynamicInfo</get>
</locationRequest>
```

```
<locationRespons>
     <dynamicInfo>
         <map>
              <name>først</name>
              <number>1</number>
              <floor>1</floor>
              <zone>
                   <name>Inngang</name>
                   <number>1</number>
                   <object>
                        <name>Statue</name>
                        <number>1</number>
                        <x>100</x>
                        <y>120</y>
                        <hs>yes</hs>
                        <url>/image/statue3</url>
                   </object>
                   <object>
                        <name>Statue 2</name>
                        <number>2</number>
                        <x>150</x>
                        <y>120</y>
                        <hs>no</hs>
                        <url>/image/statue2</url>
                   </object>
              </zone>
              <zone>
                   <name>Midtgang</name>
                   <number>2</number>
                   <floor>1</floor>
                   <located>no</located>
              </zone>
         </map>
     </dynamicInfo>
</locationResponse>
```

```
<locationRespons>
     <dynamicInfo>noinfo</dynamicInfo>
<locationRespons>
```

<get>tracks</get>

```
<locationRequest>
     <mac>00:0B:FD:C6:5C:AB</mac>
     <get>tracks</get>
</locationRequest>

<locationRequest>
     <mac>00:0B:FD:C6:5C:AB</mac>
     <get>tracks
          <name>long</name>
     </get>
</locationRequest>


<locationResponse>
     <tracks>
          <track>
               <name>long</name>
               <number>2</number>
               <coord>
                    <x1>324</x1>
                    <y1>232</y1>
                    <x2>324</x2>
                    <y2>232</y2>
                    <x3>324</x3>
                    <y3>232</y3>
                    <x4>324</x4>
                    <y4>232</y4>
                    <x5>324</x5>
                    <y5>232</y5>
                    <x6>324</x6>
                    <y6>232</y6>
              </coord>
          </track>
     </tracks>
</locationResponse>


<locationRespons>
     <tracks>noinfo</tracks>
<locationRespons>
```

Send *message*

```
<locationRequest>
     <mac>00:0B:FD:C6:5C:AB</mac>
     <send>
         <message id="1">
               <to>truls</to>
               <header>Hallo</header>
               <body>How are you?</body>
         </message>
     </send>
</locationRequest>
```

```
<locationRequest>
     <mac>00:0B:FD:C6:5C:AB</mac>
     <get>messages</get>
</locationRequest>
```

```
<locationResponse>
     <message>
          <note id="1">
               <to>me</to>
               <from>truls</from>
               <header>Hallo</header>
               <body>How are you?</body>
          </note>
          <note id="2">
               <to>me</to>
               <from>frank</from>
               <header>Hallo</header>
               <body>Do you want a break?</body>
          </note>
     </message>
</locationResponse>
```

```
<locationResponse>
     <message>noinfo</message>
</locationResponse>
```

error response

```
<locationResponse>
     <error>
          <type>unknown</type>
          <text>illegal syntax</text>
     </error>
</locationResponse>
```

Appendix G

SQL sentences

This Appendix contains the SQL sentences that are used to create the database and the tables, and the SQL sentences that are used to add the necessary rows in the database. The rows in the database are tailored to the Nidaros Cathedral.

Create the database and its tables

```
CREATE DATABASE gLocDB;
```

```
CREATE TABLE Map
(M_name varchar(70) NOT NULL PRIMARY KEY,
M_maxNorth int,
M_maxEast int,
M_minNorth int,
M_minEast int,
M_number int,
M_floor int
);
```

```
CREATE TABLE Zones
(Z_name varchar(70) NOT NULL PRIMARY KEY,
M_name varchar(70),
Z_number int,
Z_geometry GEOMETRY
);
```

```
CREATE TABLE Objects
(O_name varchar(70) NOT NULL PRIMARY KEY,
Z_name varchar(70),
O_number int,
O_hotspotarea GEOMETRY,
O_xcoord int,
O_ycoord int,
O_url varchar(150)
);

CREATE TABLE Preferences
(P_id int NOT NULL AUTO_INCREMENT PRIMARY KEY,
O_name varchar(70),
P_preference varchar(100)
);

create table Mapping
(MA_id int NOT NULL AUTO_INCREMENT primary key,
M_name varchar(70),
Ma_height int,
Ma_width int,
Ma_name varchar(70),
Ma_rotation double,
Ma_xRotate double,
Ma_yRotate double,
Ma_xOffset int,
Ma_yOffset int
);

CREATE TABLE LogLocation
(LL_id int NOT NULL AUTO_INCREMENT PRIMARY KEY,
U_id int,
LL_xpos int,
LL_ypos int,
LL_date date,
LL_time time
);

CREATE TABLE LogObject
(LO_id int NOT NULL AUTO_INCREMENT PRIMARY KEY,
O_name varchar(70),
LO_date date,
LO_time time
);

CREATE TABLE User
(U_mac varchar(50) NOT NULL PRIMARY KEY,
U_id int,
U_name varchar(100),
U_age int,
U_sex varchar(10),
U_nick varchar(50)
);

CREATE TABLE UserGroup
(UG_id int NOT NULL AUTO_INCREMENT PRIMARY KEY,
U_mac varchar(50),
G_name varchar(100)
);

CREATE TABLE UserPreferences
(UP_id int NOT NULL AUTO_INCREMENT PRIMARY KEY,
U_mac varchar(50),
UP_preference varchar(70)
);
```

Rows inserted in the database

```
INSERT INTO Mapping VALUES (NULL, ’Nidaros’, 320, 160,
’PDA’,0, 0, 0, 80, 0);

INSERT INTO Map VALUES (’Nidaros’, 13300, 6500, 0, 0,
1, 1);

INSERT INTO Zones VALUES (’Vestfronten’, ’Nidaros’, 1,
        GeomFromText(’LINESTRING(1100 2200, 5400 2200,
        5400 1000, 1100 1000)’));
INSERT INTO Zones VALUES (’Skipet’, ’Nidaros’, 2,
        GeomFromText(’LINESTRING(2300 6500, 4200 6500,
        4200 2200, 2300 2200)’));
INSERT INTO Zones VALUES (’Tverrskipet’, ’Nidaros’, 3,
        GeomFromText(’LINESTRING(1100 7500, 5400 7500,
        5400 6500, 1100 6500 )’));
INSERT INTO Zones VALUES (’Koret’, ’Nidaros’, 4,
        GeomFromText(’LINESTRING(2300 10300, 4200 10300,
        4200 7500, 2300 7500)’));
INSERT INTO Zones VALUES (’Oktogonen’, ’Nidaros’, 5,
        GeomFromText(’LINESTRING(2300 12200, 4200 12200,
        4200 10300, 2300 10300)’));

INSERT INTO Objects VALUES (’Dopefonten’, ’Koret’, 13,
        GeomFromText(’LINESTRING(3200 10300, 4000 10300,
        4000 8700, 3200 8700)’),
        172, 71, ’C:\images’);
INSERT INTO Objects VALUES (’Rosevinduet’, ’Skipet’, 5,
        GeomFromText(’LINESTRING(2500 3900, 3800 3900,
        3800 2000, 2500 2000)’),
        160, 260, ’C:\images’);
INSERT INTO Objects VALUES (’Korsalteret’, ’Skipet’, 7,
        GeomFromText(’LINESTRING(2800 6500, 3700 6500,
        3700 5500, 2800 5500)’),
        160, 180, ’C:\images’);

INSERT INTO User VALUES (’00:50:C2:1F:30:21’, 1, ’Tag’,
0,’male’, ’Tag1’);
INSERT INTO User VALUES (’00:50:C2:1F:30:79’, 1, ’Tag’,
0,’male’, ’Tag2’);
INSERT INTO User VALUES (’00:50:C2:1F:30:86’, 1, ’Tag’,
0,’male’, ’Tag3’);
INSERT INTO User VALUES (’00:0B:FD:C6:5C:AB’, 1, ’Cisco’,
0,’male’, ’Hege’);
INSERT INTO User VALUES (’00:0B:FD:5E:00:G9’, 1, ’Cisco’,
0,’male’, ’Sigurd’);
INSERT INTO User VALUES (’00:02:8A:49:1D:CD’, 1, ’Cisco’,
0,’male’, ’Ottar’);
```

```
INSERT INTO UserGroup VALUES (NULL, ’00:50:C2:1F:30:21’,
’Gruppe 1’);
INSERT INTO UserGroup VALUES (NULL, ’00:50:C2:1F:30:79’,
’Gruppe 1’);
INSERT INTO UserGroup VALUES (NULL, ’00:50:C2:1F:30:86’,
’Gruppe 1’);
INSERT INTO UserGroup VALUES (NULL, ’00:0B:FD:C6:5C:AB’,
’Gruppe 1’);
INSERT INTO UserGroup VALUES (NULL, ’00:0B:FD:5E:00:G9’,
’Gruppe 1’);
INSERT INTO UserGroup VALUES (NULL, ’00:02:8A:49:1D:CD’,
’Gruppe 1’);
```

Appendix H

Javadoc Documentation

no.ntnu.telenor.gloc.Calculations

java.lang.Object

public *Calculations*
extends Object

Title: gLoc application

Description: Class responsible for all calculations that are done in the application

H.0.1 Constructor Summary

Description
Calculations() Constructor

H.0.2 Method Summary

Returns	Description
public ArrayList	**getPositionInfo(java.awt.Point p, no.ntnu.telenor.gloc.User u)** Method used to find which Zone, Map, and object the user have.
public Point	**getScreenCoord(java.awt.Point coord, no.ntnu.telenor.gloc.Mapping m, no.ntnu.telenor.gloc.Map map)** Method which transforms a point to fit on a mobile device screen based on Map and Mapping values
public void	**print()** Method used for debugging, Prints location information
public void	**putMap(java.lang.String name, no.ntnu.telenor.gloc.Map m)** Put a map in the Hashtable

H.0.3 Constructors

Calculations

*public **Calculations()***

Constructor

H.0.4 Methods

getPositionInfo

public ArrayList ***getPositionInfo****(java.awt.Point p, no.ntnu.telenor.gloc.User u)*

Method used to find which Zone, Map, and object the user have. This is based on his location

Parameters:

p	Current location
u	Current user

Returns:

Arraylist with position information (Zone, Object, Map)

getScreenCoord

public Point ***getScreenCoord****(java.awt.Point coord, no.ntnu.telenor.gloc.Mapping m, no.ntnu.telenor.gloc.Map map)*

Method which transforms a point to fit on a mobile device screen based on Map and Mapping values

Parameters:

coord	UTM/RT90 coordinates in meters
m	Mapping to use when transforming
map	

Returns:

Transformed coordinates

print

public void ***print****()*

Method used for debugging, Prints location information

putMap

public void ***putMap****(java.lang.String name, no.ntnu.telenor.gloc.Map m)*

Put a map in the Hashtable

Parameters:

name	Map name
m	Map object

no.ntnu.telenor.gloc.Constants

java.lang.Object

---

public *Constants*
extends Object

Title: gLoc application

Description: Stores all the constants that are used by the system

H.0.5 Field Summary

Type	Description
public static final String	**DB_JDBCDRIVER** The driver to the database
public static final String	**DB_PASSWORD** The password to the database
public static final String	**DB_URL** The URL to the database
public static final String	**DB_USERNAME** The username to the database
public static final int	**HYSTERESE** Hysterese value
public static final String	**RADIOEYE_PW** The password for the RadioEye
public static final String	**RADIOEYE_URL** The URL for the RadioEye
public static final String	**RADIOEYE_USER** The user name for the RadioEye

H.0.6 Constructor Summary

Description
Constants()

H.0.7 Fields

DB_JDBCDRIVER

public static final String ***DB_JDBCDRIVER***

The driver to the database

DB_PASSWORD

public static final String ***DB_PASSWORD***

The password to the database

DB_URL

public static final String ***DB_URL***

The URL to the database

DB_USERNAME

public static final String ***DB_USERNAME***

The username to the database

HYSTERESE

public static final int ***HYSTERESE***

Hysterese value

RADIOEYE_PW

public static final String ***RADIOEYE_PW***

The password for the RadioEye

RADIOEYE_URL

public static final String ***RADIOEYE_URL***

The URL for the RadioEye

RADIOEYE_USER

public static final String ***RADIOEYE_USER***

The user name for the RadioEye

H.0.8 Constructors

Constants

public ***Constants()***

no.ntnu.telenor.gloc.CoordinatTransformer

java.lang.Object

---

public *CoordinatTransformer*
extends Object

Title: gLoc application

Description: Handles transformation of a point from UTM/RT90 to screen coordinates This class is based on the PDAScreenUTMCoord in the package ”no.ntnu.telenor.nidaros”

H.0.9 Constructor Summary

Description
CoordinatTransformer() Empty constructor
CoordinatTransformer(int maxUTME, int minUTME, int maxUTMN, int minUTMN, int width, int height, double angle, double xRot, double yRot, int tx, int ty) Constructor for setting max and min values.

H.0.10 Method Summary

Returns	Description
public Point	**getScreenCoord(java.awt.Point p)** This method translates coordinates from UTM/RT90 to a mobile device screen cooridinates Send in values in meters and get back x,y in pixels

H.0.11 Constructors

CoordinatTransformer

*public **CoordinatTransformer**()*

Empty constructor

CoordinatTransformer

*public **CoordinatTransformer**(int maxUTME, int minUTME, int maxUTMN, int minUTMN, int width, int height, double angle, double xRot, double yRot, int tx, int ty)*

Constructor for setting max and min values. Default using UTM coordinatsystem. However, it is also possible to use local coordinatsystem (RT90). Device screen size is set by width and height

Parameters:

maxUTME	Value for the maps biggest UTM easting coordinate
minUTME	Value for the maps smallest UTM easting coordinate
maxUTMN	Value for the maps biggest UTM northing coordinate
minUTMN	Value for the maps smallest UTM northing coordinate
width	The screen width in pixels
height	The screen height in pixels
angle	The angle to rotate a map
xRot	The x value to rotate around
yRot	The y value to rotate around
tx	Move the translation with x pixels in x direction
ty	Move the translation with y pixels in y direction

H.0.12 Methods

getScreenCoord

public Point ***getScreenCoord****(java.awt.Point p)*

This method translates coordinates from UTM/RT90 to a mobile device screen coorid-inates Send in values in meters and get back x,y in pixels

Parameters:

p	

Returns:

x y screencoordinates as a Point object Returns (0,0) if values are out of bound

no.ntnu.telenor.gloc.DbLocationInfo

java.lang.Object

---

public *DbLocationInfo*
extends Object

Title: gLoc application

Description: Used to get location information from the MySQL database

H.0.13 Constructor Summary

Description
DbLocationInfo() The Constructor makes a connection to the database

H.0.14 Method Summary

Returns	Description
public ResultSet	**getMappings(java.lang.String map_name)** Retrive all mappings om current map from database
public ResultSet	**getMaps()** Find all maps in db
public long	**getNumberOfZones()** Retrieves the numer of zones stored in the database
public ResultSet	**getObjectGeometry(java.lang.String z_name)** Method used to read the object geometry from db
public ResultSet	**getObjects(java.lang.String zone)** Retrieves the objects stored in the database
public ResultSet	**getZoneGeometry(java.lang.String m_name)** Method used to read zone geometry brom db
public ResultSet	**getZones(java.lang.String map)** Retrieves the zones stored in the database

H.0.15 Constructors

DbLocationInfo

*public **DbLocationInfo**()*

The Constructor makes a connection to the database

H.0.16 Methods

getMappings

*public ResultSet **getMappings**(java.lang.String map_name)*

Retrive all mappings om current map from database

Parameters:

map_name	Name on current map

Returns:

All mappings

getMaps

*public ResultSet **getMaps**()*

Find all maps in db

Returns:

Resultset with maps

getNumberOfZones

*public long **getNumberOfZones**()*

Retrieves the numer of zones stored in the database

Returns:

Number of Zones stored in the database

getObjectGeometry

*public ResultSet **getObjectGeometry**(java.lang.String z_name)*

Method used to read the object geometry from db

Parameters:

z_name	Name on zone

Returns:

A Resultset with objects geometry

getObjects

public ResultSet ***getObjects****(java.lang.String zone)*

Retrieves the objects stored in the database

Parameters:

zone	Current zone

Returns:

All the objects stored in the database as resultset

getZoneGeometry

public ResultSet ***getZoneGeometry****(java.lang.String m_name)*

Method used to read zone geometry brom db

Parameters:

m_name	Map name

Returns:

A Resultset with zone geometry

getZones

public ResultSet ***getZones****(java.lang.String map)*

Retrieves the zones stored in the database

Parameters:

map	

Returns:

All the zones stored in the database

no.ntnu.telenor.gloc.DbLog

java.lang.Object

public *DbLog*
extends Object

Title: gLoc application

Description: Used to read/write logging information to db: (Not implemented for this versjon)

H.0.17 Constructor Summary

Description
DbLog()

H.0.18 Constructors

DbLog

public ***DbLog****()*

no.ntnu.telenor.gloc.DbManager

java.lang.Object

public *DbManager*
extends Object

Title: gLoc application

Description:

H.0.19 Constructor Summary

Description
DbManager() Constructor

H.0.20 Method Summary

Returns	Description
public ResultSet	**getMappings(java.lang.String m)** Method that fetch mappings from db
public ResultSet	**getMaps()** Method that returns all available maps from db
public ResultSet	**getMyFriendsFromDb(java.lang.String mac)** Method used to get all friends to user from database
public long	**getNumberOfZones()** Method that return number of zones in db
public ResultSet	**getObjectGeometry(java.lang.String z\_name)** Method that get all of current zones object geometrys from db
public ResultSet	**getObjects(java.lang.String name)** Methods that fetches all object to current Zone
public ResultSet	**getZoneGeometry(java.lang.String m\_name)** Method that gets all of current map Zone geometrys from db
public ResultSet	**getZones(java.lang.String m\_name)** Method that fetches all zones to current map from db
public ResultSet	**initUsersFromDb()** Method to load users from db
public void	**printDbUsers()** Method to print users

H.0.21 Constructors

DbManager

public ***DbManager()***

Constructor

H.0.22 Methods

getMappings

public ResultSet ***getMappings****(java.lang.String m)*

Method that fetch mappings from db

Parameters:

m	Mapping name

Returns:

Resultset with mappings

getMaps

public ResultSet ***getMaps****()*

Method that returns all available maps from db

Returns:

ResultSet with maps

getMyFriendsFromDb

public ResultSet ***getMyFriendsFromDb****(java.lang.String mac)*

Method used to get all friends to user from database

Parameters:

mac	adress

Returns:

ResultSet rs from database

getNumberOfZones

*public long **getNumberOfZones**()*

Method that return number of zones in db

Returns:

Number of zones

getObjectGeometry

*public ResultSet **getObjectGeometry**(java.lang.String z_name)*

Method that get all of current zones object geometrys from db

Parameters:

z_name	Current zones name

Returns:

ResultsSet with object geometry

getObjects

*public ResultSet **getObjects**(java.lang.String name)*

Methods that fetches all object to current Zone

Parameters:

name	Current zone name

Returns:

Resultset with all objects in current zone

getZoneGeometry

*public ResultSet **getZoneGeometry**(java.lang.String m_name)*

Method that gets all of current map Zone geometrys from db

Parameters:

m_name	Current map name

Returns:

Resultset with Zone geometry

getZones

public ResultSet ***getZones****(java.lang.String m_name)*

Method that fetches all zones to current map from db

Parameters:

m_name	Current map name

Returns:

Resultset with zones

initUsersFromDb

public ResultSet ***initUsersFromDb****()*

Method to load users from db

Returns:

Resultset of users

printDbUsers

public void ***printDbUsers****()*

Method to print users

no.ntnu.telenor.gloc.DbUsers

java.lang.Object

public *DbUsers*
extends Object

Title: gLoc application

Description: This class is used fo fetch information about all users from db

H.0.23 Constructor Summary

Description
DbUsers() Constructor Makes connection to db

H.0.24 Method Summary

Returns	Description
public ResultSet	**getMyFriends(java.lang.String mac)** The metode takes inn a mac adress and return a resultset of mac adresses with all of the frindes to the mac adress including the incoming mac itself
public ResultSet	**initUsers()** This method reads user info from db
public void	**printUsers()** Metode to print all users in dataBase sed for debugging

H.0.25 Constructors

DbUsers

*public **DbUsers()***

Constructor Makes connection to db

H.0.26 Methods

getMyFriends

public ResultSet ***getMyFriends****(java.lang.String mac)*

The metode takes inn a mac adress and return a resultset of mac adresses with all of the frindes to the mac adress including the incoming mac itself

Parameters:

mac	Current users mac

Returns:

Resultset with friends

initUsers

public ResultSet ***initUsers()***

This method reads user info from db

Returns:

Resultset with all users

printUsers

public void ***printUsers()***

Metode to print all users in dataBase sed for debugging

no.ntnu.telenor.gloc.GLocObject

java.lang.Object

public *GLocObject*
extends Object

Title: gLoc application

Description: This Class holds Information about all objects/hotspots that are used by the system

H.0.27 Field Summary

Type	Description
public GLocObject	**next** A referens to next object

H.0.28 Constructor Summary

Description
GLocObject(java.lang.String na, int nu, java.awt.Polygon h, int xc, int yc, java.lang.String u) Constructor

H.0.29 Method Summary

Returns	Description
public String	**getName()** Method to fetch name
public int	**getNumber()** Method to fetch number
public int	**getXcoord()** Method to fetch xcoord
public int	**getYcoord()** Method to fetch ycoord
public boolean	**inside(java.awt.Point p)** Checks if a Point is inside objects area
public void	**print()** Prints info about Objects Used for debugging

H.0.30 Fields

next

*public GLocObject **next***

A referens to next object

H.0.31 Constructors

GLocObject

*public **GLocObject**(java.lang.String na, int nu, java.awt.Polygon h, int xc, int yc, java.lang.String u)*

Constructor

Parameters:

na	Name
nu	Number
h	HotSpot area
xc	x Coordinate
yc	y Coordinate
u	Objects Url

H.0.32 Methods

getName

*public String **getName**()*

Method to fetch name

Returns:

Objects name

getNumber

*public int **getNumber**()*

Method to fetch number

Returns:

Objects number

getXcoord

*public int **getXcoord**()*

Method to fetch xcoord

Returns:

Objects x coordinate

getYcoord

*public int **getYcoord**()*

Method to fetch ycoord

Returns:

Objects y coordinate

inside

*public boolean **inside**(java.awt.Point p)*

Checks if a Point is inside objects area

Parameters:

p	The point to control

Returns:

true/false depending on result

print

*public void **print**()*

Prints info about Objects Used for debugging

no.ntnu.telenor.gloc.GLocServlet

java.lang.Object

HttpServlet

public *GLocServlet*
extends HttpServlet

Title: gLoc application

Description: This is the servlet class commuicating with the mobile devices.

H.0.33 Constructor Summary

Description
GLocServlet()

H.0.34 Method Summary

Returns	**Description**
public void	**destroy()** This method ends the connection to the database
public void	**doGet(HttpServletRequest req, HttpServletResponse resp)** This method handles the request from the client, and returns a respons with proper information back to the client
public void	**doPost(HttpServletRequest req, HttpServletResponse resp)** This method handles the request from the client, and returns a respons with proper information back to the client
public void	**init()** Creates an object of class MainController

H.0.35 Constructors

GLocServlet

*public **GLocServlet()***

H.0.36 Methods

destroy

*public void **destroy**()*

This method ends the connection to the database

doGet

*public void **doGet**(HttpServletRequest req, HttpServletResponse resp)*

throws ServletException, IOException

This method handles the request from the client, and returns a respons with proper information back to the client

Parameters:

req	The request object from the client
resp	The response object to the client

Throws:

ServletException
IOException

doPost

*public void **doPost**(HttpServletRequest req, HttpServletResponse resp)*

throws ServletException, IOException

This method handles the request from the client, and returns a respons with proper information back to the client

Parameters:

req	The request object from the client
resp	The response object to the client

Throws:

ServletException
IOException

init

public void ***init****()*

throws ServletException

Creates an object of class MainController

Throws:

ServletException

no.ntnu.telenor.gloc.MainController

java.lang.Object

public *MainController*
extends Object

Title: gLoc application

Description: This Class are in charge of coordinating all information going through the system. The systems main class

H.0.37 Constructor Summary

Description
MainController() Constructor Creates an object of class DbManager, PositionManager, UserManager and XMLTransformer The constructor also initialise information from database

H.0.38 Method Summary

Returns	Description
public String	**getAreaAndPositionInfo(java.awt.Point po, java.lang.String usersMac, java.lang.String mapping, no.ntnu.telenor.gloc.User u)** This is a Method that find all location information based on users Position That is, zone, objects and map
public void	**initLocationInfo()** Fetches all location info about map,zones,objects and mappings from database The information is stored in local Java classes to be used later
public void	**initUsersFromDB()** This method initializes every user strored in the database It set ut all of it friends and other informaiton stored in the database The users are stored in the database to be used later
public String	**locationRequesetHandler(java.lang.String xmlString)** This function handles all incominc xml messages
public String	**requestHandler(java.lang.String[][] requestArray)** methode that handles all request from the 2 dim table

H.0.39 Constructors

MainController

public ***MainController()***

Constructor Creates an object of class DbManager, PositionManager, UserManager and XMLTransformer The constructor also initialise information from database

H.0.40 Methods

getAreaAndPositionInfo

public String ***getAreaAndPositionInfo****(java.awt.Point po, java.lang.String usersMac, java.lang.String mapping, no.ntnu.telenor.gloc.User u)*

This is a Method that find all location information based on users Position That is, zone, objects and map

Parameters:

po	Users position
usersMac	Current users mac
mapping	Current users mapping
u	Current users object

Returns:

Position xml ready to send to client

initLocationInfo

public void ***initLocationInfo()***

Fetches all location info about map,zones,objects and mappings from database The information is stored in local Java classes to be used later

initUsersFromDB

public void ***initUsersFromDB()***

This method initializes every user strored in the database It set ut all of it friends and other informaiton stored in the database The users are stored in the database to be used later

locationRequesetHandler

public String ***locationRequesetHandler****(java.lang.String xmlString)*

This function handles all incominc xml messages

Parameters:

xmlString	Contaioning xml format

Returns:

Answere to the request on xml format

requestHandler

*public String **requestHandler**(java.lang.String[][] requestArray)*

methode that handles all request from the 2 dim table

Parameters:

requestArray	Array containg all users requests

Returns:

locationResponse as XML

no.ntnu.telenor.gloc.Map

java.lang.Object

---

public *Map*
extends Object

Title: gLoc application

Description: Class that describs Maps used in application

H.0.41 Constructor Summary

Description
Map() Default Constructor
Map(java.lang.String n, int maxN, int maxE, int minN, int minE, int numb, int fl) Constructor to sett Map values from db

H.0.42 Method Summary

Returns	**Description**
public int	**getFloor()** Method to fetch map floor
public GLocObject	**getGlocObject(java.lang.String zonename, java.awt.Point p)** Method to find Object inside zones
public Mapping	**getMappingObject(java.lang.String mappingname)** Method that returns a mapping object based on its name
public int	**getMaxEast()** Method to fetch maps max easting
public int	**getMaxNorth()** Method to fetch maps max northing
public int	**getMinEast()** Method to fetch maps min easting
public int	**getMinNorth()** Method to fetch maps min northing
public String	**getName()** Method to fetch map name
public int	**getNumber()** Method to fetch map number
public Zone	**getZone(java.awt.Point p)** Find the zone the user are located inside

Returns	Description
public boolean	**insideZone(java.awt.Point p)** Method to find if you are inside this zone
public void	**printMappings()** Print all mappings Used for debugging
public void	**printZonesAndObjects()** Print all zone and objects Used for debugging
public void	**putMapping(no.ntnu.telenor.gloc.Mapping m, java.lang.String name)** Puts a Mapping in the mapping hashtable
public void	**putZones(no.ntnu.telenor.gloc.Zone z, java.lang.String name)** Puts the zones in an hashtable

H.0.43 Constructors

Map

*public **Map**()*

Default Constructor

Map

*public **Map**(java.lang.String n, int maxN, int maxE, int minN, int minE, int numb, int fl)*

Constructor to sett Map values from db

Parameters:

n	Map name
maxN	Maximum northing
maxE	Maximum easting
minN	Minimum northing
minE	Minimum easting
numb	Map number
fl	Floor number

H.0.44 Methods

getFloor

*public int **getFloor**()*

Method to fetch map floor

Returns:

Map floor

getGlocObject

public GLocObject ***getGlocObject****(java.lang.String zonename, java.awt.Point p)*

Method to find Object inside zones

Parameters:

zonename	Zone name
p	Users location

Returns:

Current Object

getMappingObject

public Mapping ***getMappingObject****(java.lang.String mappingname)*

Method that returns a mapping object based on its name

Parameters:

mappingname	Mapping Name

Returns:

Mapping object

getMaxEast

public int ***getMaxEast****()*

Method to fetch maps max easting

Returns:

Max easting value

getMaxNorth

public int ***getMaxNorth()***

Method to fetch maps max northing

Returns:

Max norhting value

getMinEast

public int ***getMinEast()***

Method to fetch maps min easting

Returns:

Min easting value

getMinNorth

public int ***getMinNorth()***

Method to fetch maps min northing

Returns:

Min norhting value

getName

public String ***getName()***

Method to fetch map name

Returns:

Map name

getNumber

public int ***getNumber()***

Method to fetch map number

Returns:

Map number

getZone

*public Zone **getZone**(java.awt.Point p)*

Find the zone the user are located inside

Parameters:

p	Current location

Returns:

Zone object

insideZone

*public boolean **insideZone**(java.awt.Point p)*

Method to find if you are inside this zone

Parameters:

p	Current location

Returns:

true/false base on inside or not

printMappings

*public void **printMappings**()*

Print all mappings Used for debugging

printZonesAndObjects

*public void **printZonesAndObjects**()*

Print all zone and objects Used for debugging

putMapping

*public void **putMapping**(no.ntnu.telenor.gloc.Mapping m, java.lang.String name)*

Puts a Mapping in the mapping hashtable

Parameters:

m	
name	

putZones

public void ***putZones****(no.ntnu.telenor.gloc.Zone z, java.lang.String name)*

Puts the zones in an hashtable

Parameters:

z	Zone object
name	Zone name

no.ntnu.telenor.gloc.Mapping

java.lang.Object

public *Mapping*
extends Object

Title: gLoc application

Description: This Class descibes the mapping fra World(UTM/RT90) coordinate to map on mobile devices screen.

H.0.45 Constructor Summary

Description
Mapping() Default Constructor
Mapping(java.lang.String m, int mh, int mw, int mox, int moy, double mr, double xRot, double yRot) This constructor sets all mapping values

H.0.46 Method Summary

Returns	**Description**
public int	**getMappingHeigth()** Return heigth of mobile devices map in pixel
public String	**getMappingName()** Return current mapping name
public int	**getMappingOffsetX()** Return pixels to move screen position in x direction
public int	**getMappingOffsetY()** Return pixels to move screen position in y direction
public double	**getMappingRotation()** Return rotation angle in radians.
public int	**getMappingWidth()** Return width of mobile devices map in pixel
public double	**getMappingxRotate()** The x coordinate to rotate around
public double	**getMappingyRotate()** The y coordinate to rotate around

H.0.47 Constructors

Mapping

*public **Mapping()***

Default Constructor

Mapping

*public **Mapping**(java.lang.String m, int mh, int mw, int mox, int moy, double mr, double xRot, double yRot)*

This constructor sets all mapping values

Parameters:

m	Mapping name
mh	Mapping heigth
mw	Mapping width
mox	Mapping offset x
moy	Mapping offset y
mr	Mapping rotation angle
xRot	Mapping rotation x point
yRot	Mapping rotation y point

H.0.48 Methods

getMappingHeigth

*public int **getMappingHeigth**()*

Return heigth of mobile devices map in pixel

Returns:

Mapping height

getMappingName

*public String **getMappingName**()*

Return current mapping name

Returns:

Mapping name

getMappingOffsetX

*public int **getMappingOffsetX()***

Return pixels to move screen position in x direction

Returns:

Mapping offsetx

getMappingOffsetY

*public int **getMappingOffsetY()***

Return pixels to move screen position in y direction

Returns:

Mapping offsety

getMappingRotation

*public double **getMappingRotation()***

Return rotation angle in radians. Is used to rotate screen coordinates

Returns:

Rotation angle

getMappingWidth

*public int **getMappingWidth()***

Return width of mobile devices map in pixel

Returns:

Mapping width

getMappingxRotate

*public double **getMappingxRotate()***

The x coordinate to rotate around

Returns:

xRotate

getMappingyRotate

public double ***getMappingyRotate****()*

The y coordinate to rotate around

Returns:

yRotate

no.ntnu.telenor.gloc.MPPContentHandler

java.lang.Object

org.xml.sax.helpers.DefaultHandler

---

public *MPPContentHandler*
extends DefaultHandler

Title: gLoc application

Description: Validates and extracts the different elements from the RadioEye XML response according to the appropriate dtd type definition file. (Class is modyfied to be used with new Cordis RadioEye db)

H.0.49 Field Summary

Type	Description
public static final String[]	**MPP4TAG**

H.0.50 Constructor Summary

Description
MPPContentHandler() Creates a new Position object

H.0.51 Method Summary

Returns	Description
public void	**characters(char[] ch, int start, int length)** Receive notification of character data.
public void	**endElement(java.lang.String namespaceURI, java.lang.String localName, java.lang.String qName)** Receive notification of the end of an element.
public Position	**getPosition()** Method which returns a reference to a Position object
public InputSource	**resolveEntity(java.lang.String publicId, java.lang.String systemId)** Resolve an external entity.

Returns	Description
public void	**startElement(java.lang.String namespaceURI, java.lang.String localName, java.lang.String qName, org.xml.sax.Attributes atts)** Receive notification of the beginning of an element.

H.0.52 Fields

MPP4TAG

public static final String[] ***MPP4TAG***

H.0.53 Constructors

MPPContentHandler

public ***MPPContentHandler****()*

Creates a new Position object

H.0.54 Methods

characters

public void ***characters****(char[] ch, int start, int length)*

throws SAXException

Receive notification of character data.

The Parser will call this method to report each chunk of character data. SAX parsers may return all contiguous character data in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity so that the Locator provides useful information.

Parameters:

ch	The characters from the XML document
start	The start position in the array.
length	The number of characters to read from the array.

Throws:

SAXException	Any SAX exception, possibly wrapping another exception.

endElement

*public void **endElement**(java.lang.String namespaceURI, java.lang.String localName, java.lang.String qName)*

throws SAXException

Receive notification of the end of an element.

The SAX parser will invoke this method at the end of every element in the XML document; there will be a corresponding startElement event for every endElement event (even when the element is empty).

Parameters:

namespaceURI	The Namespace URI, or the empty string if the elementhas no Namespace URI or if Namespace processing is not being performed.
localName	The local name (without prefix), or the empty string if Namespace processing is not being performed.
qName	The qualified XML 1.0 name (with prefix), or the empty string if qualified names are not available.

Throws:

SAXException	Any SAX exception, possibly wrapping another exception.

getPosition

*public Position **getPosition**()*

Method which returns a reference to a Position object

Returns:

position a reference to a Position object

resolveEntity

*public InputSource **resolveEntity**(java.lang.String publicId, java.lang.String systemId)*

throws SAXException

Resolve an external entity. In this case finding the correct document type defenition (dtd) for validating the XML data.

Parameters:

publicId	The public identifer, or null if none is available.
systemId	The system identifier provided in the XML document.

Returns:

InputSource The new input source, or null to require the default behaviour.

Throws:

SAXException	Any SAX exception, possibly wrapping another exception

startElement

*public void **startElement**(java.lang.String namespaceURI, java.lang.String localName, java.lang.String qName, org.xml.sax.Attributes atts)*

throws SAXException

Receive notification of the beginning of an element.

The Parser will invoke this method at the beginning of every element in the XML document; there will be a corresponding endElement event for every startElement event (even when the element is empty). All of the element's content will be reported, in order, before the corresponding endElement event

Parameters:

namespaceURI	The Namespace URI, or the empty string if the element has no Namespace URI or if Namespace processing is not being performed.
localName	The local name (without prefix), or the empty string if Namespace processing is not being performed.
qName	The qualified name (with prefix), or the empty string if qualified names are not available.
atts	The attributes attached to the element. If there are noattributes, it shall be an empty Attributes object.

Throws:

SAXException	Any SAX exception, possibly wrapping another exception.

no.ntnu.telenor.gloc.Position

java.lang.Object

public *Position*
extends Object
implements Serializable

Title: gLoc application

Description: This class holds all information about current position

H.0.55 Constructor Summary

Description
Position()

H.0.56 Method Summary

Returns	Description
public int	**getAltitude()** Returns the altitude of the wireless device or the RadioEye position.
public String	**getDateTimeMs()** Returns the dateTimeMs timestamp in ms.
public int	**getLatitudeUTM()** Returns the latitude (EASTING) UTM value
public int	**getLongitudeUTM()** Returns the longitude (NORTHING) UTM value
public double	**getxRT90()** Returns the x value from the local coordinatsystem RT-90
public double	**getyRT90()** Returns the y value from the local coordinatsystem RT-90
public void	**setAltitude(int altitude)** Sets the altitude of the wireless device or the RadioEye position.
public void	**setDateTimeMs(java.lang.String dateTimeMs)** Sets the date and time in ms.
public void	**setLatitudeUTM(int latitudeUTM)** Sets the latitud (EASTING) UTM value
public void	**setLongitudeUTM(int longitudeUTM)** Sets the longitude (NORTHING) UTM value
public void	**setxRT90(double x)** Sets the xRT90 value

Returns	Description
public void	**setyRT90(double y)** Sets the yRT90 value

H.0.57 Constructors

Position

*public **Position()***

H.0.58 Methods

getAltitude

*public int **getAltitude()***

Returns the altitude of the wireless device or the RadioEye position.

Returns:

The altitude

getDateTimeMs

*public String **getDateTimeMs()***

Returns the dateTimeMs timestamp in ms.

Returns:

dateTimeMs The timestamp in ms.

getLatitudeUTM

*public int **getLatitudeUTM()***

Returns the latitude (EASTING) UTM value

Returns:

the latitude in UTM value

getLongitudeUTM

*public int **getLongitudeUTM()***

Returns the longitude (NORTHING) UTM value

Returns:

the longitude in UTM value

getxRT90

*public double **getxRT90()***

Returns the x value from the local coordinatsystem RT-90

Returns:

x value to RT-90

getyRT90

*public double **getyRT90()***

Returns the y value from the local coordinatsystem RT-90

Returns:

y value as RT-90 value

setAltitude

*public void **setAltitude**(int altitude)*

Sets the altitude of the wireless device or the RadioEye position.

Parameters:

altitude	The altitude

setDateTimeMs

*public void **setDateTimeMs**(java.lang.String dateTimeMs)*

Sets the date and time in ms.

Parameters:

dateTimeMs	The timestamp in ms.

setLatitudeUTM

public void ***setLatitudeUTM****(int latitudeUTM)*

Sets the latitud (EASTING) UTM value

Parameters:

latitudeUTM	the latitude in UTM value

setLongitudeUTM

public void ***setLongitudeUTM****(int longitudeUTM)*

Sets the longitude (NORTHING) UTM value

Parameters:

longitudeUTM	the longitude in UTM value

setxRT90

public void ***setxRT90****(double x)*

Sets the xRT90 value

Parameters:

x	from RadioEye

setyRT90

public void ***setyRT90****(double y)*

Sets the yRT90 value

Parameters:

y	from RadioEye

no.ntnu.telenor.gloc.PositionGetterRE

java.lang.Object

public *PositionGetterRE*
extends Object

Title: gLoc application

Description: Gets positioning information from the Cordis RadioEye and stores relevant data for the given MAC address in a Position object for later use. The location is returned as an XML response from the RadioEye's web server.

H.0.59 Constructor Summary

Description
PositionGetterRE() Constructor

H.0.60 Method Summary

Returns	Description
public Position	**getPositionRE(java.lang.String mac)** Sending a HTTPrequest to the RadioEye asking for the position to a wireless device idenitified by a MAC address.

H.0.61 Constructors

PositionGetterRE

*public **PositionGetterRE**()*

Constructor

H.0.62 Methods

getPositionRE

*public Position **getPositionRE**(java.lang.String mac)*

Sending a HTTPrequest to the RadioEye asking for the position to a wireless device idenitified by a MAC address.

Parameters:

mac	MAC address of the requested device.

Returns:

Position object containing the found position or null if position data is unavailable

no.ntnu.telenor.gloc.PositionManager

java.lang.Object

public *PositionManager*
extends Object

Title: gLoc application

Description: This Class manage all informaiton concerning position

H.0.63 Constructor Summary

Description
PositionManager() Constructor

H.0.64 Method Summary

Returns	Description
public int[]	**getPosition(java.lang.String mac)** Method for finding users position in meters
public ArrayList	**getPositionInfo(java.awt.Point p, no.ntnu.telenor.gloc.User u)** Find Postion information
public Point	**getScreenCoordinates(java.awt.Point c, no.ntnu.telenor.gloc.Mapping m, no.ntnu.telenor.gloc.Map map)** Transform Point to screen coordinates
public void	**print()** Method used for debugging, Prints location information
public void	**putMap(no.ntnu.telenor.gloc.Map m, java.lang.String name)** Method to put map to Calculations map table

H.0.65 Constructors

PositionManager

public ***PositionManager()***

Constructor

H.0.66 Methods

getPosition

public int[] ***getPosition****(java.lang.String mac)*

Method for finding users position in meters

Parameters:

mac	The mac adress the device that are beeing positioned

Returns:

Current users postion

getPositionInfo

public ArrayList ***getPositionInfo****(java.awt.Point p, no.ntnu.telenor.gloc.User u)*

Find Postion information

Parameters:

p	Current postion in meters
u	Current user

Returns:

Arraylist with postion info

getScreenCoordinates

public Point ***getScreenCoordinates****(java.awt.Point c, no.ntnu.telenor.gloc.Mapping m, no.ntnu.telenor.gloc.Map map)*

Transform Point to screen coordinates

Parameters:

c	Current locaition in meters
m	Current mapping
map	Current Map

Returns:

Transformed coordinates

print

*public void **print**()*

Method used for debugging, Prints location information

putMap

*public void **putMap**(no.ntnu.telenor.gloc.Map m, java.lang.String name)*

Method to put map to Calculations map table

Parameters:

m	Map objcet
name	

no.ntnu.telenor.gloc.PositionSimulator

java.lang.Object

public *PositionSimulator*
extends Object

Title: gLoc application

Description: Class for simulating users position under testing of Client applications

H.0.67 Constructor Summary

Description
PositionSimulator() Constructor

H.0.68 Method Summary

Returns	Description
public int[]	**getPosition(int startX, int startY, int stopX, int stopY, int stepX, int stepY)** This methoded returns simulated values.

H.0.69 Constructors

PositionSimulator

*public **PositionSimulator**()*

Constructor

H.0.70 Methods

getPosition

*public int[] **getPosition**(int startX, int startY, int stopX, int stopY, int stepX, int stepY)*

This methoded returns simulated values. The users will move with speed stepY and stepX in a straight line from start to stop. Simulator will be restarted when start or stop values are changed or stop values are reached

Parameters:

startX	Starting x point
startY	Starting y point
stopX	Stopping x point
stopY	Stopping y point
stepX	Step in meters in x direction
stepY	Step in meters in y direction

Returns:

New simulated value

no.ntnu.telenor.gloc.TestMain

java.lang.Object

public *TestMain*
extends Object

Title: gLoc application

Description: Class for unit testing of the Runtime System during the development phase

H.0.71 Constructor Summary

Description
TestMain()

H.0.72 Method Summary

Returns	Description
public static void	**main(java.lang.String[] a)** The main class starts the test by sending a locationRequest to the Main-controller All information is then written to a console window to monitor that it is easy to monitor

H.0.73 Constructors

TestMain

public ***TestMain****()*

H.0.74 Methods

main

public static void ***main****(java.lang.String[] a)*

throws InterruptedException

The main class starts the test by sending a locationRequest to the Maincontroller All information is then written to a console window to monitor that it is easy to monitor

Parameters:

a	Nothing

Throws:

InterruptedException

no.ntnu.telenor.gloc.User

java.lang.Object

public *User*
extends Object

Title: gLoc application

Description: This class contain information about a user

H.0.75 Constructor Summary

Description
User() Empty constructor
User(java.lang.String m, java.lang.String n, java.lang.String nic, java.lang.String s, int a) Constructor which sets all the variables

H.0.76 Method Summary

Returns	**Description**
public String	**getMac()** Used to get the users objects mac adress
public String	**getMapName()** This method returns users current map
public int	**getMapNumber()** Return users last map number
public ArrayList	**getMyFriends()** Returns all the registered friends of an user
public ArrayList	**getMyMessages()** Recieves all the stored messages to the user
public String	**getName()** Used to get the user objects name
public String	**getNick()** Used to get users nick
public Point	**getPosition()** This method return users position.
public Zone	**getPrevZone()** Return users last zone

Returns	Description
public int[]	**getUsersSimulatedPosition(int startX, int startY, int stopX, int stopY, int stepX, int stepY)** Method that return simulated position
public boolean	**isMyFriend(java.lang.String friend)** Find friends
public void	**putFriend(java.lang.String friend)** Adds a friend to the friendslist
public void	**putMessage(java.lang.String message)** Stores the message to the user
public void	**putPosition(java.awt.Point p, no.ntnu.telenor.gloc.Zone z)** Sets the last position the user is located at
public void	**setMapInfo(java.lang.String name, int number)** Sets map name and number

H.0.77 Constructors

User

*public **User**()*

Empty constructor

User

*public **User**(java.lang.String m, java.lang.String n, java.lang.String nic, java.lang.String s, int a)*

Constructor which sets all the variables

Parameters:

m	The mac adress of the handheld device the user is using
n	The user’s name
nic	The users’s nick
s	The user’s sex
a	The user’s age

H.0.78 Methods

getMac

*public String **getMac**()*

Used to get the users objects mac adress

Returns:

An String with user mac

getMapName

public String ***getMapName()***

This method returns users current map

Returns:

Map name

getMapNumber

public int ***getMapNumber()***

Return users last map number

Returns:

Map number

getMyFriends

public ArrayList ***getMyFriends()***

Returns all the registered friends of an user

Returns:

An arraylist of all the friends

getMyMessages

public ArrayList ***getMyMessages()***

Recieves all the stored messages to the user

Returns:

An arrayList with all the messages

getName

*public String **getName()***

Used to get the user objects name

Returns:

An String with user name

getNick

*public String **getNick()***

Used to get users nick

Returns:

users nick

getPosition

*public Point **getPosition()***

This method return users position. Returns (0,0) if not set

Returns:

p Position

getPrevZone

*public Zone **getPrevZone()***

Return users last zone

Returns:

Zone name

getUsersSimulatedPosition

*public int[] **getUsersSimulatedPosition**(int startX, int startY, int stopX, int stopY, int stepX, int stepY)*

Method that return simulated position

Parameters:

startX	Decribe where the simulation starts

startY	Decribe where the simulation starts
stopX	Decribe where the simulation stops
stopY	Decribe where the simulation stops
stepX	Length of each step
stepY	Length of each step

Returns:

int[] with new position

isMyFriend

*public boolean **isMyFriend**(java.lang.String friend)*

Find friends

Parameters:

friend	The mac address or nick of the friend

Returns:

True if the requested user is registered as a friend. else, false.

putFriend

*public void **putFriend**(java.lang.String friend)*

Adds a friend to the friendslist

Parameters:

friend	

putMessage

*public void **putMessage**(java.lang.String message)*

Stores the message to the user

Parameters:

message	The message to the user

putPosition

*public void **putPosition**(java.awt.Point p, no.ntnu.telenor.gloc.Zone z)*

Sets the last position the user is located at

Parameters:

p	The last position the user is located at
z	

setMapInfo

*public void **setMapInfo**(java.lang.String name, int number)*

Sets map name and number

Parameters:

name	Current maps name
number	Current maps number

no.ntnu.telenor.gloc.UserManager

java.lang.Object

public *UserManager*
extends Object

Title: gLoc application

Description: This class manages User and Friend information

H.0.79 Constructor Summary

Description
UserManager() Constructor

H.0.80 Method Summary

Returns	Description
public void	**addMyFriendsFromDB(java.sql.ResultSet rs, java.lang.String mac)** This metod takes in one users mac and a resultsett with all its friend the friend is then added to the user object
public void	**adduser(java.lang.String mac)** Adds a user to the Hashtable with all the users.
public void	**addUserPostion(java.lang.String mac, java.awt.Point p, no.ntnu.telenor.gloc.Zone z)** This updates users position
public void	**addusersFromDB(java.sql.ResultSet rs)** Add users to the user Hashtable.
public Hashtable	**getAllUsers()** Returns the user table
public int[]	**getSimulatedPosition(java.lang.String mac, int startX, int startY, int stopX, int stopY, int stepX, int stepY)** Method used to find simulated position
public User	**getUser(java.lang.String usersMac)** Return current user based on mac adress
public Hashtable	**getUserFriends(java.lang.String mac)** Find all friends to current user
public void	**setMapInfo(java.lang.String mac, java.lang.String mapname, int mapnumber)** Set Users map info

H.0.81 Constructors

UserManager

*public **UserManager**()*

Constructor

H.0.82 Methods

addMyFriendsFromDB

*public void **addMyFriendsFromDB**(java.sql.ResultSet rs, java.lang.String mac)*

This metod takes in one users mac and a resultsett with all its friend the friend is then added to the user object

Parameters:

rs	The Result from friends in database
mac	The mac for the user

adduser

*public void **adduser**(java.lang.String mac)*

Adds a user to the Hashtable with all the users. The mac address is the key

Parameters:

mac	The mac adress to the user

addUserPostion

*public void **addUserPostion**(java.lang.String mac, java.awt.Point p, no.ntnu.telenor.gloc.Zone z)*

This updates users position

Parameters:

mac	Users mac adress
p	Users location
z	Users current zone

addusersFromDB

*public void **addusersFromDB**(java.sql.ResultSet rs)*

Add users to the user Hashtable. The mac address is the key

Parameters:

rs	

getAllUsers

*public Hashtable **getAllUsers**()*

Returns the user table

Returns:

allUsers: The Hashtable where users are stored

getSimulatedPosition

*public int[] **getSimulatedPosition**(java.lang.String mac, int startX, int startY, int stopX, int stopY, int stepX, int stepY)*

Method used to find simulated position

Parameters:

mac	Current users mac
startX	Start value
startY	Start value
stopX	Stop value
stopY	Stop value
stepX	Step value
stepY	Step value

Returns:

Simulated location

getUser

*public User **getUser**(java.lang.String usersMac)*

Return current user based on mac adress

Parameters:

usersMac	Current users mac

Returns:

User Object

getUserFriends

*public Hashtable **getUserFriends**(java.lang.String mac)*

Find all friends to current user

Parameters:

mac	Current users mac adress

Returns:

A hashtable with friends

setMapInfo

*public void **setMapInfo**(java.lang.String mac, java.lang.String mapname, int mapnumber)*

Set Users map info

Parameters:

mac	Users mac
mapname	Current map name
mapnumber	Current map number

no.ntnu.telenor.gloc.XMLTransformer

java.lang.Object

public *XMLTransformer*
extends Object

Title: gLoc application

Description: Class used to transforme incoming and outgoing XML data

H.0.83 Constructor Summary

Description
XMLTransformer() Constructor

H.0.84 Method Summary

Returns	Description
public String	**addXMLStrings(java.lang.String xml_1, java.lang.String xml_2)** Add 2 XML strings together
public String	**getFinaleXML(java.lang.String xml)** Method that returns the final response XML
public String	**getFriendsPositionXML(java.util.Hashtable friends)** This method take in current users friend position information and return friendPostion XML
public String	**getPositionXML(java.awt.Point p, java.lang.String mac, java.lang.String mapping, no.ntnu.telenor.gloc.Map m, no.ntnu.telenor.gloc.Mapping mapp, no.ntnu.telenor.gloc.Zone z, no.ntnu.telenor.gloc.GLocObject globj)** This method takes in all of current users position information found by the system and convert it to response XML
public String[][]	**getUserRequest(java.lang.String xmlString)** This method is used to find all request that is going to be handled by the system it is stored in a 2 dim table and sent back

H.0.85 Constructors

XMLTransformer

*public **XMLTransformer()***

Constructor

H.0.86 Methods

addXMLStrings

*public String **addXMLStrings**(java.lang.String xml_1, java.lang.String xml_2)*

Add 2 XML strings together

Parameters:

xml_1	String number 1
xml_2	String number 2

Returns:

A XML string

getFinaleXML

*public String **getFinaleXML**(java.lang.String xml)*

Method that returns the final response XML

Parameters:

xml	XML string corresponding to positon

Returns:

locationResponse XML

getFriendsPositionXML

*public String **getFriendsPositionXML**(java.util.Hashtable friends)*

This method take in current users friend position information and return friendPostion XML

Parameters:

friends	Table with all friends

Returns:

friendPosition response XML

getPositionXML

*public String **getPositionXML**(java.awt.Point p, java.lang.String mac, java.lang.String mapping, no.ntnu.telenor.gloc.Map m, no.ntnu.telenor.gloc.Mapping mapp, no.ntnu.telenor.gloc.Zone z, no.ntnu.telenor.gloc.GLocObject globj)*

This method takes in all of current users position information found by the system and convert it to response XML

Parameters:

p	Users position
mac	Users mac adresss
mapping	Users mapping
m	Users Map
mapp	Users Mapping object
z	Users Zone object
globj	Users GLoacObject

Returns:

position response XML

getUserRequest

*public String[][] **getUserRequest**(java.lang.String xmlString)*

This method is used to find all request that is going to be handled by the system it is stored in a 2 dim table and sent back

Parameters:

xmlString	Users XML request

Returns:

Users XML request transformed to a 2 dimentional array

no.ntnu.telenor.gloc.Zone

java.lang.Object

public *Zone*
extends Object

Title: gLoc application

Description: The Zone class holdes information about a zone inside a map

H.0.87 Constructor Summary

Description
Zone(java.lang.String zname, int n, java.awt.Polygon g, java.awt.Polygon h) Constructor

H.0.88 Method Summary

Returns	Description
public void	**addObject(no.ntnu.telenor.gloc.GLocObject o)** Adds a object to object list
public GLocObject	**getGLocObject(java.awt.Point p)** Find current GLocObject that match location
public String	**getName()** Method to return zone name
public int	**getNumber()** Method to return zone number
public boolean	**inside(java.awt.Point p)** Test if point is inside zone
public boolean	**insideHysterese(java.awt.Point p)** Test if location are inside Hystorese Geometry
public void	**printObjects()** Print all objects to current zone Used for debugging
public void	**printZone()** Print zone information Used for debugging

H.0.89 Constructors

Zone

*public **Zone**(java.lang.String zname, int n, java.awt.Polygon g, java.awt.Polygon h)*

Constructor

Parameters:

zname	Zone name
n	Zone number
g	Zone Geometry
h	Zone Hystorese Geometry

H.0.90 Methods

addObject

*public void **addObject**(no.ntnu.telenor.gloc.GLocObject o)*

Adds a object to object list

Parameters:

o	New Object

getGLocObject

*public GLocObject **getGLocObject**(java.awt.Point p)*

Find current GLocObject that match location

Parameters:

p	Current location

Returns:

GLocObject

getName

*public String **getName**()*

Method to return zone name

Returns:

Zone Name

getNumber

*public int **getNumber**()*

Method to return zone number

Returns:

Zone number

inside

*public boolean **inside**(java.awt.Point p)*

Test if point is inside zone

Parameters:

p	Current location

Returns:

true if inside/false if not

insideHysterese

*public boolean **insideHysterese**(java.awt.Point p)*

Test if location are inside Hystorese Geometry

Parameters:

p	Current location

Returns:

true if inside/false if outside

printObjects

*public void **printObjects**()*

Print all objects to current zone Used for debugging

printZone

public void ***printZone()***

Print zone information Used for debugging