Master Thesis 2004

Micro Positioning

-tracking objects, finding objects and getting alarms-

Solrun Furuholt Pedersen

TRONDHEIM JUNE 15, 2004

Department of Telematics
Faculty for Information Technology, Mathematics and Electrical Engineering
Norwegian University of Science and Technology

For
Bravida Geomatikk AS
Trondheim
Norway

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY, MATHEMATICS AND ELECTRICAL ENGINEERING

MASTER THESIS

Student's name:	Solrun Furuholt Pedersen
Field of study:	Telematics
Title:	**Micro positioning**
Description:	To obtain the position of an object may be useful in many situations. Positioning a person by using the GPS or GSM technology has been possible for some time. These methods do however only have accuracy up to about 50-300 meters at the best. Micro positioning is a new concept which can provide an accuracy of around 1 meter, and can be as precise as 1 cm with certain techniques. This thesis aims at designing a practical system where micro positioning is used. The environment for the system shall be a hospital. In a hospital it might be useful to track objects (for instance medical equipment and beds), find objects (for instance doctors, nurses, patients), and get alarms. Alarms can be triggered if a mental patient tries to walk out of his ward, if a TV is moved out of a certain room, and so on. In all of these areas, obtaining the position of the objects is critical.
Start date:	January 20, 2004
Deadline:	June 15, 2004
Submission date:	
Department:	Department of Telematics
Supervisor:	Steinar Høseggen, Bravida Geomatikk AS

Trondheim, January 20, 2004

Rolv Bræk
Professor

Preface

This thesis is carried out as a part of the MSc studies at the Norwegian University of Science and Technology, Faculty of Information Technology, Mathematics and Electrical Engineering, Department of Telematics. It is the result of the master thesis 2004 in the field of Systems Engineering.

The thesis is conducted as a collaboration between NTNU and Bravida Geomatikk AS. It has partly been developed at NTNU, Department of Telematics, and partly at Bravida Geomatikk's facilities at Tyholt.

I would in particular like to thank my mentor at Bravida Geomatikk, Steinar Høseggen, for invaluable help and guidance. During meetings he has helped me to get a better understanding of the system to be specified.

I also want to thank my mentor at NTNU, Professor Rolv Bræk, for guidance and support in writing the report.

Other people that deserve thanks are my co-workers at Bravida Geomatikk, and Ingebrigt Fuglem at Telenor FoU. They have all provided technical assistance during the work.

The last thanks go to Håvard Sjøvoll for useful hints in the development process, and for help reading through the thesis.

Trondheim June 15, 2004.

Solrun Furuholt Pedersen

Abstract

To obtain the position of an object by using the GSM or the GPS technology has been possible for quite some time. The accuracy provided by these methods is however generally low, in the range of 200 meters for the GSM technology, and around 30 meters for GPS. Lately there has emerged a need for more accurate positioning techniques that can be used indoors. Micro positioning is a term often used for new concepts, which can provide an accuracy of around 1 meter, and can be as precise as 1 cm with certain techniques.

This thesis is a collaboration between NTNU and Bravida Geomatikk AS. It aimed at specifying a system where the micro positioning concepts could be used. A hospital should be used to illustrate an environment where such a system could be employed. The main objective of the thesis was to find the requirements applying to the system, and to design it. A small demonstrator implementing some basic concepts should also be developed.

The background for developing such a system is the needs arising from large buildings, like hospitals. In a hospital it might be useful to track objects, for instance medical equipment and beds, to find persons, for instance doctors, nurses and patients, and to get alarms. Alarms can be triggered if a mental patient tries to walk out of his ward or if a TV is moved out of a certain room. In all of these areas, obtaining the position of the objects is critical.
Micro positioning concepts can also be used to offer location aware services to persons at the hospital, as well as providing facility management support.

To solve the task, a prestudy was first carried out. It introduced the system to be designed, and investigated application areas where it could be used. The concepts of positioning were explored, and an overview of the existing equipment to be used for micro positioning was given. Other systems that would surround the system to be designed were also studied.
A requirements specification for the system was then developed, and based on this, the functional design for the system was given. This design considers all the major concepts of the system, except those relating to non-functional requirements. From this design, a small demonstrator was made, implementing a limited behaviour of the full system. The development of the demonstrator proved successful. It has been carefully tested, and worked as intended.

Where's my trousers gone?
Where's my best shirt gone?
Where's the mouth organ then with only four notes in?
Where is Jesper's hat?
Where is this and that?
Where's the brand new leather purse with very few notes in?
I can say I had it with me yesterday.

From 'The robber's looking song'.
Original words and music: Thorbjørn Egner
English words: Anthony Barnett

Contents

Figure list

Table list

Abbreviations and explanations

This section will provide explanations for abbreviations and other terms frequently used in this thesis.

3D - positioning	*Three Dimensional Positioning*. A technique where the location of an object is given in 3 coordinates: longitude, latitude and altitude. I.e. the x-, y-, and z- coordinates are used
API	*Application Program Interface*. A set of routines, protocols, and tools for building software applications. A good API makes it easier to develop a program by providing the building blocks. A programmer puts the blocks together
GIS	*Geographic Information System*. GIS is a computer-based information system that enables capture, modelling, manipulation, retrieval, analysis and presentation of geographically referenced data. A GIS is an overall system, and not just a static database. Two of the most important technologies underlying GIS are database systems and computer graphics. It is a system that can handle objects (OI's) that have a geographic location and/or a geographic area, and that has some kind of information related to it. Examples of such information could be a product description, a status description, etc
GPS	*Global Positioning System*. A worldwide satellite navigational system formed by 24 satellites orbiting the earth, and their corresponding receivers on the earth
GSM	*Global System for Mobile communication*. GSM service is available in more than 100 countries and has become the de facto standard for digital cellular systems in Europe and Asia
GUI	*Graphical User Interface.* A program interface that takes advantage of the computer's graphics capabilities to make the program easier to use. Well-designed graphical user interfaces can free the user from learning complex command languages. Microsoft Windows is an example of a GUI
HTML	*Hyper Text Markup Language*. The language used to create documents for the World Wide Web
HTTP	*Hyper Text Transfer Protocol.* The underlying protocol used by the World Wide Web. It defines how messages are formatted and transmitted
IDE	*Integrated Development Environment.* A programming environment integrated into an application. For example, Microsoft Office applications support various versions of the BASIC programming language. You can develop a WordBasic application while running Microsoft Word
IEEE	*Institute of Electrical and Electronics Engineers*. IEEE is an organization composed of engineers, scientists, and students, and was founded in 1884. The IEEE is best known for developing standards for the computer and electronics industry. In particular, the IEEE 802 standards for local-area networks are widely followed
IP	*Internet Protocol.* Specifies the format of packets, also called datagrams, and the addressing scheme of the Internet
IPS	*Indoor Positioning System.* A typical IPS tracks people wearing badges or carrying devices that transmit wireless signals to receivers in ceilings or on walls. The receivers, connected to a local network, send the data to servers that calculate location and make the information available in various ways
JAR	*Java Archive*. A file format used to bundle all components required by a Java application. JAR files simplify the downloading of applications since all the components (.class files, images, sounds, etc.) can be packaged into a single file. In addition, JAR supports data compression, which further decreases download times.

JAVA	Java is a high-level programming language developed by Sun Microsystems. It is an object-oriented language similar to C++, but simplified to eliminate language features that cause common programming errors. Java source code files (files with a .java extension) are compiled into a format called bytecode (files with a .class extension), which can then be executed by a Java interpreter. Compiled Java code can run on most computers because Java interpreters and runtime environments, known as Java Virtual Machines
JDBC	*Java DataBase Connectivity*. A Java API that enables Java programs to execute SQL statements. This allows Java programs to interact with any SQL-compliant database. Since nearly all relational database management systems (DBMSs) support SQL, and because Java itself runs on most platforms, JDBC makes it possible to write a single database application that can run on different platforms and interact with different DBMSs
JMS	*Java Message Service*. The JMS API is a messaging standard that allows application components based on the Java 2 Platform, Enterprise Edition (J2EE) to create, send, receive, and read messages. It enables distributed communication that is loosely coupled, reliable, and asynchronous
JNDI	*Java Naming and Directory Interface*. A part of the Java platform, providing applications based on Java technology with a unified interface to multiple naming and directory services
JSP	*Java Server Pages*. A server-side technology. Java Server Pages is an extension to the Java servlet technology that was developed by SUN.
Laptop	A small, portable computer. It's small enough to be sitting on a persons lap
LAN	*Local Area Network*. A computer network that spans a relatively small area. Most LANs are confined to a single building or group of buildings
LBS	*Location Based Services*. The ability to find the geographical location of the mobile device and provide services based on this location information
MSC	*Message Sequence Chart*. A tool for specification and validation, which concentrates on describing the message-sending between instances
OI	*Object of Interest*. A term frequently used in the GIS world. It is used to mean any objects that is of interest to a system. It is possible to position it, and to get information about it
PDA	*Personal Digital Assistant*. A handheld device that combines computing, telephone/fax, Internet and networking features.
RFID	*Radio Frequency Identification*. A technology similar in theory to bar code identification. An RFID system consists of an antenna and a transceiver, which read the radio frequency and transfer the information to a processing device, and a transponder, or tag, which is an integrated circuit containing the information to be transmitted
SAPT	*System for Alarm, Positioning and Tracking*. The abbreviation that has been set on the system that is to be designed in this thesis
SDL	*Specification and Description Language*. A language used to give telecommunication administrations and manufacturers a common language for precise and unambiguous communication about the behaviour of telecommunications systems
SQL	*Structured Query Language*. A standardized query language for requesting information from a database. SQL was first introduced as a commercial database system in 1979 by Oracle Corporation
Tablet PC	A type of notebook computer that has an LCD screen on which the user can write using a special-purpose pen. The handwriting is digitized and can be converted to standard text through handwriting recognition, or it can remain as handwritten text. Tablet PCs also typically have a keyboard and/or a mouse for input.

Tag	In this thesis, a tag is used to mean a device that an object will need to possess in order to be able to be positioned. The device will be a sort of a transmitter/receiver that can transmit location information for the object to the system
TCP	*Transmission Control Protocol*. One of the main protocols in TCP/IP networks. Whereas the IP protocol deals only with packets, TCP enables two hosts to establish a connection and exchange streams of data. TCP guarantees delivery of data and also guarantees that packets will be delivered in the same order in which they were sent
UML	*Unified Modelling Language*. A general-purpose notational language for specifying and visualizing complex software, especially large, object-oriented projects
URL	*Uniform Resource Locator*. The global address of documents and other resources on the World Wide Web. The first part of the address indicates what protocol to use, and the second part specifies the IP address or the domain name where the resource is located
Wi-Fi	*Wireless Fidelity*. Used generically when referring of any type of 802.11 network, whether 802.11b, 802.11a, dual-band, etc
WLAN	*Wireless Local Area Network.* A type of local-area network that uses high-frequency radio waves rather than wires to communicate between nodes

Explanations are mainly quoted from [1].

1 Introduction

'Location, location, location' is no longer a truism limited to real estate agents. It has become the mantra of a growing number of companies as they seek new ways to track objects.

Location Based Services (LBS) are touted as the 'new killer application' of the mobile information revolution. According to [2], LBS is *the ability to find the geographical location of a mobile device, and provide services based on this location information.*

1.1 Background

To obtain the position of an object by using the GSM or the GPS technology has been possible for quite some time. The accuracy provided by these methods is however generally low, in the range of 200 meters for the GSM technology, and around 30 meters for GPS. Lately there has emerged a need for more accurate positioning techniques that can be used indoors. Micro positioning is a term often used for new concepts, which can provide an accuracy of around 1 meter, and can be as precise as 1 cm with certain techniques.

This thesis will describe a location based service where one uses the location of objects to be able to track them down easily. The environment for the service will be a hospital. In a hospital it might be useful to track objects, for instance medical equipment and beds, to find persons, for instance doctors, nurses and patients, and to get alarms. Alarms can be triggered if a mental patient tries to walk out of his ward or if expensive equipment is moved out of a certain room. In all of these areas, obtaining the position of the objects is critical.

1.2 Objective

The main objective of this thesis is to find the requirements applying to a system for alarm, positioning and tracking, and to design the system. A small demonstrator implementing important concepts shall also be made.
The environment for the system will be St. Olav's Hospital, which is situated in the city of Trondheim, Norway.

To be able to reach this objective, a prestudy of the concept needs to be performed. This will involve investigating application areas for the system, along with related work. A study of different techniques and methods that can be used for micro positioning is also required.

Special challenges relates to the following areas:

- to find the requirements that applies to the system, both functional and non-functional
- to design the system functionally, in accordance with the requirements
- to handle interfaces between the systems
- to make the different components communicate with each other in an efficient way

1.3 Assumptions and limitations

This study is written in collaboration with Bravida Geomatikk AS in Trondheim, Norway. They have given the outline for the study, and defined the task. The basis for this thesis is as shown in figure 3–1 on page 9. What shall be designed in this thesis is this system, with special focus on the 'top system' (system for alarm, positioning and tracking). The other components in the figure shall not be designed or implemented. It shall be assumed that they already exist. They shall just be studied in order to understand them, to be able to integrate them in the overall solution.
A thing to note is that the thesis shall just *study* techniques for micro positioning. It shall not find new techniques for it. It shall rather explore how different technologies for this can be integrated in a hospital environment. Different technologies might be used side by side for a best possible solution.
Because of limited time available, the non-functional aspect of the system will not be emphasized.

1.4 The report

Since this is a master thesis in the field of systems engineering, the report takes a systems engineering point of view. A large section is assigned to the requirements specification, as well as to the design and implementation processes. Less space is given to explain background technologies used, for instance GSM and the Internet. This is assumed to be known to the reader.

The thesis is divided in two main parts: chapters 2-6 provide some background material needed to be able to understand the system that is to be developed, while chapters 7-11 describes the specific work that has been done.

The thesis starts with a brief introduction to some terms that will be frequently used in the rest of the report. Chapter 3 then presents a prestudy that was done in the beginning of the process to get a better understanding of the system that was to be developed. It gives an impression of how the current situation at a hospital is today, and application areas where the new system can be used. Chapter 4 studies different techniques that can be used for positioning. Special attention is paid to techniques that support micro positioning. Chapter 5 looks at the

surroundings for the system that is to be designed. Chapter 6 describes the methodology used in the rest of the report, and gives a brief revisit to the purpose of the task.

The second part of the thesis starts with a requirement specification, giving both functional and non-functional requirements that apply to the system. Chapter 8 gives the functional design that has been made for the study, while chapter 9 describes a partly implementation design for the system. Chapter 10 evaluates a prototype for the system that was made in 2003, while chapter 11 describes the implementation done for this thesis.

Chapter 12 gives ideas for future work that can be done to improve the system, while chapter 13 finishes of with a conclusion.

2 Positioning and Objects of Interest

This chapter will describe what is meant by the terms 'positioning and 'micro positioning'. It will also introduce the term 'Object of Interest'.
These terms are frequently used in the rest of this report.

2.1 Positioning

To position an object means to obtain the location of the object. A key factor is that the object that is to be positioned must contain some sort of a transmitter/receiver that is communicating with one or more gateways/antennas. Typically, the transmitter/receiver can be a GPS device, a mobile phone, a PC with WLAN capabilities, etc. The gateway/antenna can be a satellite, a GSM base station, a WLAN access point, etc.

Positioning can be implemented in two ways: self positioning and remote positioning. These two approaches are described in section 2.1.1 and section 2.1.2.
Section 2.1.3 and section 2.1.4 specify the difference between outdoor and indoor (micro) positioning.

2.1.1 Self-positioning

This is a way to implement positioning where the transmitter/receiver uses signals communicated by gateways/antennas to calculate its own position. More specifically, the positioning transmitter/receiver makes the appropriate signal measurements from geographically distributed gateways/antennas and uses these measurements to determine its position. A self-positioning transmitter/receiver, therefore, ‘knows’ where it is and applications collocated with the transmitter/receiver can use this information to make positioned-based decisions. [3]
This technique is used by the GPS system.

2.1.2 Remote positioning

In this case, the transmitter/receiver can be located by measuring the signals travelling to and from a set of gateways/antennas. More specifically, the gateways/antennas, which can be

installed at one or more locations, measure a signal originating from, or reflecting off, the object to be positioned. These signal measurements are used to determine the length and/or direction of the individual radio paths, and then the transmitter/receiver position is computed from geometric relationships. [3]
This technique is used in both WLAN- and ultrasound positioning.

2.1.3 Outdoor positioning

The term ‘outdoor positioning’ is used here to denote positioning of an object by using equipment installed outdoors. This can for instance be GSM antennas or GPS satellites. Techniques for outdoor positioning often have limitations such that they cannot be used indoors. For instance, GPS signals require an unbroken line of sight, and thus cannot penetrate walls. I.e. it is best suited for outdoor use. Other techniques are not accurate enough to be used indoors. An example is GSM positioning, where the accuracy can range from 100 meters to 35km.

2.1.4 Indoor positioning

Indoor positioning, on the other side, refers to obtaining the location of objects at a smaller scale. Naturally, techniques for this will need to be more accurate than outdoor positioning techniques. It is of little interest to position an object with accuracy of 100 meters indoor. Typically, for higher accuracy, more infrastructure is needed. The gateways/antennas mentioned earlier needs to be spaced more densely, and other techniques than used outdoor usually have to be employed. From now on, the term 'micro positioning' will be used to indicate indoor positioning. The precision of a micro positioning system is usually down to one or a few meters, but can also be as precise as ±1 cm with certain 3D techniques. It is micro positioning that will be the focus of this study.

2.2 Object of Interest

The term ‘Object of Interest’, or just ‘OI’, is often used in the GIS (Geographic Information System) word to denote any objects that are of interest to a system. In the system depicted in this study, examples of OIs can be medical equipment, beds, staff, patients and the cancer department.
It will typically be possible to obtain the position of an OI, and to retrieve information about it. All details of an OI will be stored in a database, and the positions of an OI will be available through maps.

An OI can be either static or dynamic. A static OI has information that relates to a given geographical area, while a dynamic OI also contains information that includes a given period of time. An example of a static OI could be a table, and an example of a dynamic OI could be a doctor.
The spatio-temporal aspect gives an extra dimension, as an OI can be perceived differently depending on the current time. For instance, on Monday the 10th of March from 8am to 4pm, Maria Hansen is the nurse in charge for the cancer department. However, from 4pm to 8am the next morning, Carl Pettersen holds this position. [4]

In this thesis the term ‘OI’ and the term ‘object’ has the same meaning.

3 Prestudy

As a first step towards solving the task, a prestudy was carried out.
It started by studying the current routines at a real hospital. This revealed many interesting aspects, and gave ideas to what could be done to improve the current status. The structure that would be needed to realize the depicted system was also outlined. Technology and components that could be used for micro positioning were explored, and these are described in chapter 4: 'Techniques for positioning'. Finally, the surroundings for the system were studied in detail. Special attention was paid to the interfaces that these provided. This is described in chapter 5: 'Surrounding systems'.

3.1 Situation at St. Olav's Hospital today

The prestudy started with an investigation of the situation at a hospital today. The hospital studied was St. Olav's Hospital in Trondheim, Norway. Particular attention was paid to the areas where the use of micro positioning might come in handy. It was discovered that a lot of the routines in these areas were ineffective, and performance could be increased with the use of a system like the one depicted in this study.

From [5] and [6] it was found that:

- It is not a very widespread use of information technology amongst the staff at the hospital.
- There exists no system to find lost things. If, for instance, some medical equipment is lost, a manual search has to be performed.
- Staff are mainly found by the use of pagers, and sometimes phone calls.
- The pager system does not have coverage at all parts of the hospital.
- Staff that are currently not at the hospital's premises are hard to localize.
- When an alarm goes off, it can be hard to localize its exact location.
- If a mental patient disappears, a manual search for the patient has to be performed.
- There is poor control with the location of visitors.
- There is poor control with the management of equipment/beds/etc. When some beds are worn-out, all beds bought at the same time are changed. This even though some of the beds might have been at a storeroom the whole time, and has almost never been used.

- Equipment at the hospital is quite easy to steal. For instance, at one of the wards at the hospital, four TVs disappeared last year. Some people just walked in to the hospital, and carried out a TV. One of the TVs even disappeared along with its table!

Clearly, there exist potential for improvement here, and that is exactly what this thesis will try to do something about.

3.2 Introduction to the new system

The new system that is to be made is built around the logical architecture shown in figure 3–1 on page 9, and the document described in [7]. This is what was given as a basis to work from by the proposer of this task. A brief description of each of the components is given in the following.

Users of the system will typically be nurses, doctors, hospital orderlies, etc. In the figure it is shown that these users can access the system through a stationary PC, but this can be any terminal connected to the web, for instance a laptop, a PDA, etc.
The 'System for Alarm, Positioning and Tracking' (from now denoted ‘SAPT’) component is the main component that connects the other parts together. It is also the application that serves users of the system. The Object Database contains information for all objects (OIs) that it is possible to obtain locations for. This can be both mobile and stationary objects. The Map System contains maps of the area of interest, i.e. the hospital. The Map System will present maps in the scale that the user requests, to graphically show the location of objects. The Message Service is a separate system that is used by the SAPT to send messages to other systems. The Catalogue Service is also a separate system. It contains information about all objects at the hospital. This can for instance be phone number, address, etc., for all staff, and serial numbers, purchase date, user manuals, etc., for equipment.
The Location Server is a subsystem that fetches new positions for objects. The collection of positions (geographic coordinates) can be done with a number of different position techniques, for instance WLAN, ultrasound or GSM.
All of these components will be described in detail in the next chapters.

The main goal of this thesis is to define the functionality of the SAPT component, and describe how it can be realized. The other systems either already exist in some form, or are to be built soon.

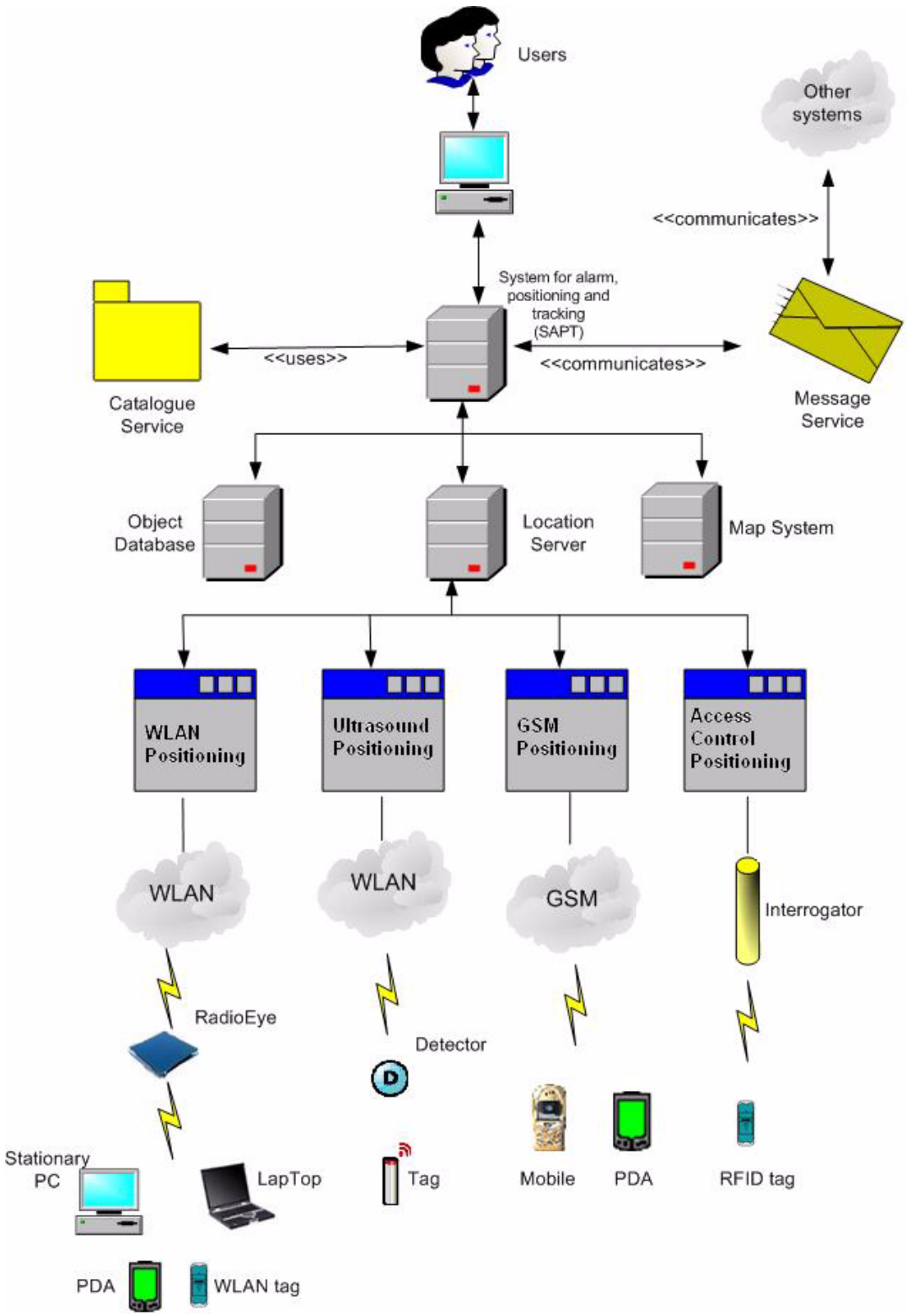


Figure 3–1: Logical architecture of the new system

3.3 Application areas

In this section, a description of different application areas for the concept will be given. This is to give the reader a perception of how the new system can be used in practice.
Hopefully the value that the system can yield to the users will be understood.

3.3.1 Tracking and finding

There are often experienced problems in finding desired equipment. The equipment could have been moved from its original place and left somewhere else. Also finding a special person can be hard in a large building. By the use of the SAPT, equipment and persons can be found easily. The SAPT will, based of information from the Location Server and information stored in the Object Database, be able to track equipment and find persons.

Example 1: A room is missing one of the beds that are supposed to be there. A nurse is wondering where it is. She can then log in to any PC, start the SAPT, and send a request asking for the bed. She quickly gets back a response telling where the bed is, a map showing its position.
Example 2: A nurse is wondering where the closest bed is. She sends a request to the SAPT, including the room number where she presently is. A response is returned, along with a map showing the location of the bed. The response can also contain information such as the bed's status (free/occupied/broken/etc.), how long it is free before somebody has reserved it, etc.
Example 3: A mental patient is missing from his room. By the use of the SAPT, the location of the patient can be obtained.

3.3.2 Securing equipment and getting alarms

Equipment has a history of disappearing without leaving a trace. Sometimes the equipment has accidentally been taken to another hospital after an emergency or transfer of a patient. Other times equipment disappears because of theft. In the SAPT, the hospital can be divided in different areas or zones. Equipment can then be registered to belong to a certain zone, and be moved freely around in this area. If it is moved out of its zone, an alarm will go off. The same principle can be used for persons.

Example 1: A TV is defined to belong to room 123. If the TV is moved around inside the room, for instance to another corner, no alarm is triggered. But as soon as the TV leaves the room, an alarm goes off.
Example 2: A mental patient can walk freely around inside his ward, but as soon as he tries to leave it, an alarm is triggered.
Example 3: A visitor can walk freely around in common area, but as soon as he tries to walk in to the staff's premises, an alarm is triggered.

3.3.3 Facility Management

This is a topic that will affect many things, ranging from outdoor areas, buildings, rooms, equipment and other properties that the hospital owns.
The SAPT is well suited to support facility management because the Object Database contains information about all objects at the hospital, including buildings.

Example 1: Information about buildings can be recorded by the SAPT. This can be information about when maintenance was last done, planned maintenance, defects that needs to be fixed, etc.
Example 2: All equipment that exists at the hospital is registered in the system. Maintenance routines are recorded, and one can use this to make sure that service intervals are followed. If some equipment is damaged, this information can be also be recorded.
Example 3: Documentation for all equipment can be stored in the system. This can be working instructions, user manuals and other necessary documents. By having all documentation collected at the same place, it is easy to check how old a document is, if a new and more up-to-date document needs to be written, who did last revise this document, etc. It is also easy to determine if one is reading the latest version of the document.

3.3.4 Location aware services

Location aware services is an interesting application area. It adapts content and service execution to a user's current physical location. This area will not receive any special attention in this thesis, but the SAPT may be extended at a later instant to cover such services as well.

Example 1: A doctor carries a mobile terminal (for instance a PDA or a tablet PC) on his daily walk around to his patients. As he approaches the different patients, the terminal automatically displays their medical record.
Example 2: A nurse carries a mobile terminal. If she stands close to some medical equipment, the terminal automatically displays the user manual for the equipment.

3.3.5 Other application fields

The system to be designed in this thesis has its focus on a hospital environment, and the examples given in this section were taken from there.
It is however worth noting that the system can be used in any big building, or number of buildings, that needs to keep track of objects and staff, and to get alarms.

Examples of other fields where the system can be employed:

- Schools, for instance universities and colleges
- Large companies
- Museums
- Storehouses and warehouses
- Large shops
- Etc. The list is infinite

That the system can be used in other applications as well, and not just at one specific hospital, increases the value of the work done in this thesis.

3.4 Related work

Several companies are developing solutions based on micro positioning concepts. As micro positioning is a relatively new idea, most of the solutions are still only prototypes, and not fully commercial systems.
Some of the numerous solutions are presented below.

Radianse, a company situated in Massachusetts, USA, has set up an IPS (indoor positioning system) at Massachusetts General Hospital in Boston. The IPS is integrated in a project for 'the Operating Room of the Future' (ORF). The system enables the hospital to track surgeons, nurses, patients and equipment during surgical operations, and to store this information in databases. The data helps researchers analyse whether the devices and processes being tested are more efficient than those used in conventional operating rooms. In addition, web browsers can be used to search for objects with tags, and to view diagrams of floors showing real-time locations of doctors, patients and equipment. [8]

Another indoor system, developed by Versus Technology of Michigan, USA, supports a wide range of healthcare IPS applications. The Versus Information System (VIS) is a real-time locating system that uses patented locating technology to instantly locate people and equipment moving within a facility. VIS, marketed primarily at the healthcare industry, takes the guesswork out of finding people and equipment, while improving efficiencies, streamline procedures, and save money. It helps the staff manage traffic more efficiently and quickly shift resources as bottlenecks or problems develop. The applications have been installed at several locations. One of them is in the busy emergency department at Albert Einstein Medical Centre in Philadelphia. [9]

At the Metro Group Future Store, a futuristic supermarket in Rheinberg, Germany, a Wi-Fi-based 'personal shopping assistant' has been developed with software from Ekahau, a California company. Using tablet PCs affixed to the front of shopping carts, shoppers can find their way to groceries they need, learn about sale items and obtain product information. [10]

Another Ekahau-based system, in use at the University of Oulu in Finland, allows students with PDAs to locate over 250,000 books at the main library. They can request for map-based guidance to a desired book or collection. [11]

Sonitor, a Norwegian company, has set up an IPS at Rikshospitalet University Hospital in Oslo. The system, called MediTRAC, helps the hospital utilize medical equipment better. It keeps records of patterns of usage for the equipment, as well as to trace their movement throughout the hospital. [12]

Another Norwegian company, Radionor, has installed its technology at Statoil's research lab in Trondheim. A 3D-model of the lab has been constructed, and it is possible to show the location of all WLAN units currently in the lab on the model. A service has also been made, which makes it possible to get information on different research projects as one walks through the lab. The content on laptops and PDAs changes as one move around. [13]

Radionor has also installed a mobile service at Nidarosdomen in Trondheim. The service is a guiding-service, which knows the exact location of a person walking inside the Nidarosdomen. A voice tells the person what he sees, while images and facts are rolling over the display as he moves around the building. [14]

4 Techniques for positioning

This chapter will study different techniques that can be used for micro positioning. There exist many different techniques for this, and there is not enough space to cover all of them in this report, so there has been made a selection. The selection is based on guidelines from the proposer of this thesis.
The GPS and GSM systems are presented as an addition, because these are the position systems known to most people. The reader will be given an understanding of why they are not suitable to be used for micro positioning.

4.1 Global Positioning System (GPS)

GPS is a worldwide satellite-based radio navigation system, consisting of 24 satellites, equally spaced in six orbital planes 20 200 kilometres above the Earth. It is a self-positioning technique. The GPS system went into effect in the early 1980s mainly for use by the US military. In May 2000 the US military eventually ceased adding deliberate error to the GPS signal for civilian receivers, and overnight, civilian GPS's became 10 times more accurate, and in many cases even better. It is now used in many commercial applications.

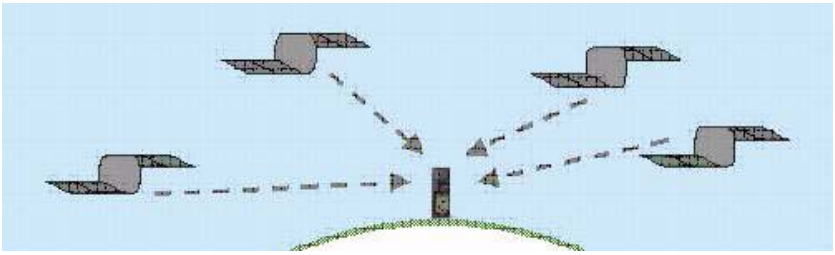


Figure 4–1: GPS receiver communicating with GPS satellites

A GPS receiver requires an unbroken line of sight to three or more of the 24 orbiting GPS satellites to attain a positional fix (figure 4–1). Each satellite contains two sets of info that it broadcasts regularly. These info sets are called the ‘almanac’ and the ‘ephemeris’. The almanac

is the rough position of all the satellites in the sky. It is broadcast by all satellites, and is valid for about 3 months. The ephemeris is the exact location of one satellite, and is only broadcast by that one satellite. This info is only valid for a few hours. In other words, each satellite broadcasts the rough position of all other satellites and the exact position of itself. The receiver keeps in its memory the most recent almanac and ephemeris data.

To obtain its position, a receiver finds a satellite in the sky and gets its broadcast. Based on the time it takes the signal from the satellite to reach the receiver, the receiver knows that it is located somewhere on a large circle. Another satellite is found and another circle can then be drawn. Now the receiver knows it is at one of the two points where the circles intersect. A third satellite is found, another circle drawn, and now it knows were it is in two dimensions within a few meters. If a fourth satellite is found, the receiver gets to know its altitude also. So it takes four satellite signals to get a 3D fix - latitude, longitude and altitude.

The Global Positioning System claims a positional accuracy of 30 metres on the ground, 95 percent of the time; however the 5 percent of predictions that are wrong by more than 30 metres are likely to occur in the places where there is most need for commercial systems, that is, in the cities.

The main advantage of the technique is that the GPS system has already been in use for many years, and is a well-tried solution. However, since the GPS receivers requires unbroken line of sight to three or more of the 24 orbiting GPS satellites, they are unable to work indoors or in vehicles, unless placed close to a window. Frequently the ‘urban canyons‘ created by buildings on either side of a street can be enough to prevent any estimate being made. It can also often take several minutes for the positional information from each satellite to be received cleanly at start-up.
To include the GPS technology in small handheld terminals is not trivial either. The addition of a GPS chip and bulky aerial will increase the power requirements and size of devices that handset manufacturers are trying to make smaller and more efficient.
[3], [15], [16].

4.2 GSM positioning

The following description requires a basic knowledge of the GSM technology. If the reader is unfamiliar with GSM, please consult [17] or [18] for additional information.

The positioning of a GSM phone can be done in several different ways [19]. The simplest method, which will be described here, involves knowing in which cell the phone currently is. Positioning of phones within a cell is a well established method, as it is required for call routing. The main benefit of the technology is that it is already in use today, and can be supported by all mobile handsets. One does not have to include any extra technology in the phones, PDA's or other handsets connected to the GSM network.
The accuracy of this technique is however generally low, in the range of 200 meters, depending on the cell size. Position accuracy is usually most precise where GPS performs worst; in urban areas where the dense network of cells limits the potential area where a phone can be. The considerably larger cells found in rural areas will give significantly poorer results. See figure 4–

2, and note the hierarchy of cells, the smallest found in the urban areas, and becoming larger in increasingly rural environments.

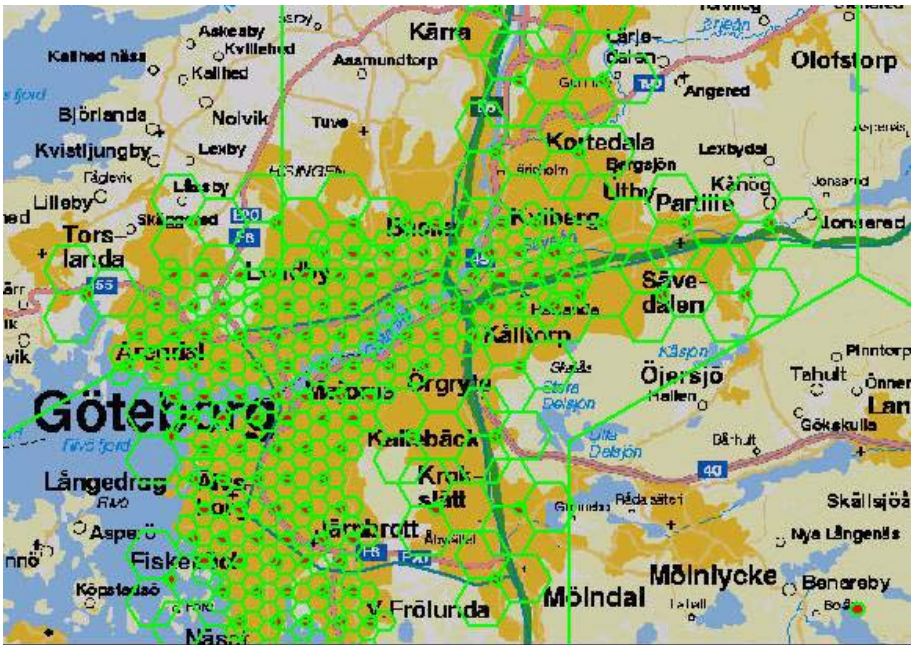


Figure 4–2: The concept of GSM cells

The technique can currently not be used for 3D positioning, as it only gives the area of cells in x-y coordinates. In addition, it can just work in places where the mobile terminal is able to get a signal.[3],[7].

Mobile phones can only be positioned (legally) if the owner has given his written permission.

4.3 WLAN positioning

WLAN positioning is a collective term used to denote techniques that is based on signals from the IEEE 802.11 standard. Many terminals can transmit information over this network, including standard PCs, PDAs, tablet PCs, etc. This section will first introduce the 802.11 standard. Following, two specific technologies that use WLAN positioning will be presented.

4.3.1 The IEEE 802.11 standard

802.11 refer to a family of specifications developed by the IEEE for wireless LAN technology (WLAN). 802.11 specify an over-the-air interface between a wireless client and a base station, or between two wireless clients. The IEEE accepted the specification in 1997.

There are several specifications in the 802.11 family:

- ***802.11*:** Applies to wireless LANs and provides 1 or 2 Mbps transmission in the 2.4 GHz.
- ***802.11a*:** An extension to 802.11 that applies to wireless LANs and provides up to 54 Mbps in the 5GHz band.

- ***802.11b*:** (also referred to as 802.11 High Rate or Wi-Fi). An extension to 802.11 that applies to wireless LANs and provides 11 Mbps transmission (with a fallback to 5.5, 2 and 1 Mbps) in the 2.4 GHz band. 802.11b was a 1999 ratification to the original 802.11 standard, allowing wireless functionality comparable to Ethernet.
- ***802.11g*:** Applies to wireless LANs and provides 20+ Mbps in the 2.4 GHz band.

[1]

4.3.2 WLAN positioning with Radionor

Radionor Communications was founded in June 2000, and is located in the city of Trondheim, Norway. The company's patented core technology is based on innovative methods for precise localization of wireless terminals inside Local Area Networks (LANs).
The system is cell based (just like GSM), and can locate objects connected to the local network via WLAN. Separate instruments called Cordis RadioEye listens for signals from the WLAN.

4.3.2.1 Cordis RadioEye (CRE)

Cordis RadioEye is a hardware based sensor capable of accurately compute the physical location coordinates for wireless network terminals. It is a stand-alone Wireless LAN-conforming (IEEE 802.11b) hardware sensor that is capable to decode the MAC addresses of network user terminals and determine their geographic position with a typical accuracy of better than ± 1 m. This is done by analysing the characteristics of the emitted microwave signal spectrum from the user terminals. Specially developed algorithms increase the confidence interval of the calculated localisation data. Only one sensor is needed to generate a position vector (2 axes) towards the terminal. The total system will typically be based on several sensors that overlap in area, and this creates redundancy and precision in localisation. The current RadioEye measures 25×25×7 cm, and one sensor may cover a floor area of up to 5000m^2.

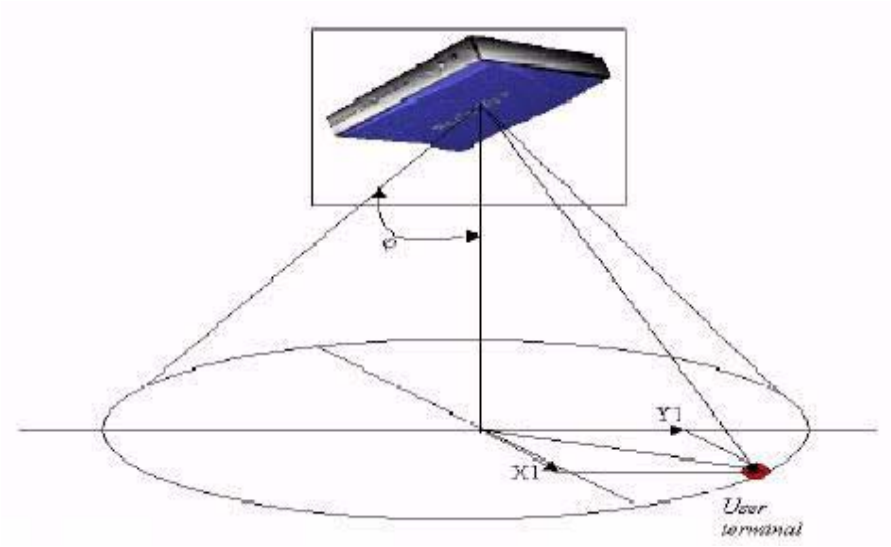


Figure 4–3: Cordis RadioEye mounted in ceiling

The product is typically mounted in the ceiling (see figure 4–3) or on walls, and will be totally separated from the wireless LAN data chains. The location of objects is done real-time, and the RadioEye can handle up to 1000 location updates per second.

No additional software or hardware is needed in terminals that are to be positioned, as the system is totally independent of the infrastructure layout. This means that all Wi-Fi devices and terminals that are used in the area can be identified and positioned. The RadioEye needs to be calibrated when installed, but does not need to be calibrated later if changes occur in the WLAN-infrastructure or in the buildings. The system adapts automatically to a changing environment.
The system is organized as an independent part of the network structure, and data from the sensors can be communicated either over WLAN (the sensors have built-in WLAN interface) or over wired Ethernet 10/100.
If there are 1000 objects that are possible to position in the same WLAN segment, the RadioEye will generate a load on the 802.11b network of about 3-4%.

If one uses two RadioEyes that covers the same area, it is possible to obtain 3D positioning, i.e. both x-, y- and z-coordinates.
[20], [21]

4.3.2.2 Radionor's WLAN tag

Objects that are not naturally connected to WLAN, for instance beds, medical equipment etc., will need a separate WLAN tag to be able to be positioned.

Radionor's WLAN tag has integrated battery with an estimated lifetime of the battery of 5,8 years (2000 transactions per day). It has integrated antenna, and communicates via IEEE 802.11b. Its range is 150 meters. The tag has an analogue sensor input for detection of motion/ shock, and digital activation input for detection of pushing an alarm button. Other inputs are possible if needed. The tag also has a memory unit for logging purposes.

The tag sends short, periodic messages on the wireless network, and these messages are received by the WLAN access points. At the same time the position of the tag is determined by the Cordis RadioEye. The periodic message contains information about the tag's identity, as well as logged data from the sensor inputs. If an alarm is triggered, a message is sent immediately via the WLAN access points, containing the tag's identity, all logs and the activator status.[21]

The evolution of tags is however rapidly evolving, and Radionor is currently working on developing smaller, lighter and better tags.

4.3.3 WLAN positioning with Ekahau

Ekahau Inc. is a US based corporation with its headquarters located in Saratoga, California. Devices that Ekahau can track include PDAs, laptops, Wi-FI tags and other 802.11 enabled wireless appliances.
Ekahau's solution is software-based only, and the main component of the solution is Ekahau Position Engine (EPE).

4.3.3.1 Ekahau Position Engine (EPE)

EPE is a software-only position server that features up to 1 meter average position accuracy. It consists of three components; Ekahau Client, Ekahau Manager and Ekahau Positioning Engine Server (see figure 4–4).

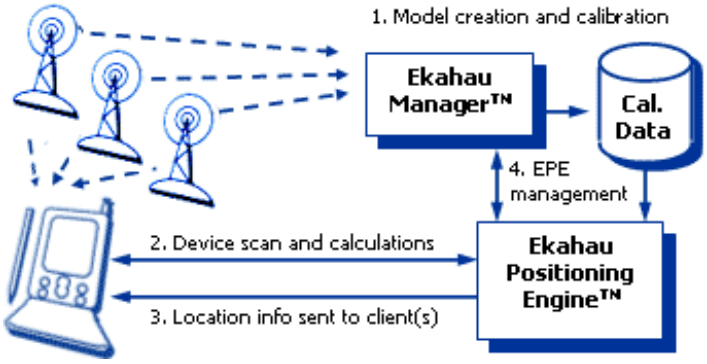


Figure 4–4: Positioning with Ekahau Position Engine

No extra physical infrastructure is needed, but new software needs to be installed in each client that is to be positioned. This software is called the Ekahau Client. It is Java-based, which makes it platform and network independent, and fits in most PCs/telephones/PDAs/etc.
The Ekahau Client is based on signal strength site calibration. The positions are calculated based on the object's signal strength from one or more WLAN access points. This method has to be calibrated at installation of the system, and each time changes occur in the WLAN infrastructure or in the buildings. The calibration data is stored in the system (Cal. Data on figure 4–4). This is done using the Ekahau Manager. Ekahau has estimated the approximate time needed for site calibration to 1 hour per 1200 m^2.

The location information is displayed in the form of x, y, floor, speed, heading, and logical area, such as 'Main Conference Room' for instance. The accuracy of the system can be up to 1 meter. If this accuracy is needed, one can add 'dummy' access points (low-cost 802.11 access points with the power on but not connected to the network) in selected site locations. For 1 meter average accuracy one needs 4-6 access point signals, and for up to 2-3 meter average accuracy one needs 1-3 signals. [22]

Currently, the Ekahau solution only works with WLAN enabled devices. They have however recently hooked up with National Scientific Corporation (NSC) to develop a WLAN tag, aimed at indoor use, which can be integrated in their solution. [23]

4.4 Ultrasound positioning

Ultrasound has several desirable properties. It does not penetrate walls, so location per room is easy to achieve. On the other hand, it does not require line of sight between the tags and the detector, making it possible to track objects that are hidden or located in drawers or filing cabinets. Ultrasound waves are mechanical waves and are immune to interference. They do not interfere with sensitive equipment that might otherwise be disturbed by electromagnetic waves.

The security of an ultrasound system is very high and it is impossible to eavesdrop on the communications link from outside the premises of the installation.
The rest of this section will take a look at a particular system that uses ultrasound for positioning.

4.4.1 Ultrasound positioning with Sonitor

Sonitor Technologies AS was founded in 1997. It is situated in the city of Oslo, Norway. Sonitor was founded as a response to the ever-growing problem of managing clinical paperwork, quality assurance of medical equipment, and especially moveable equipment used in direct contact with the patients in hospitals.
The basic idea of Sonitor Technologies AS is to apply new technology and methods to manage medical records and equipment.

4.4.1.1 Sonitor IPS ultrasound positioning

Sonitor IPS (Indoor Positioning System) is a method for positioning of objects and persons that are carrying a Sonitor Tag. The tags are sending out their ID-number over an ultrasonic link. These tag-signals are received by Sonitor Detectors that are connected to the local area network via WLAN. The detectors are mounted on the walls or ceiling in the rooms one wish to cover by the system (See figure 4–5).

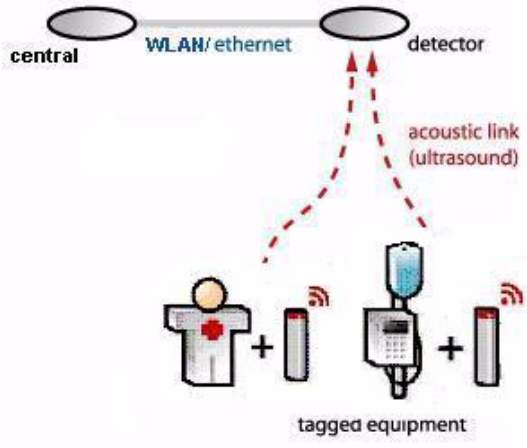


Figure 4–5: Ultrasound positioning with Sonitor

Ultrasound is restricted by solid walls, windows, etc., which makes it easy to position an object on room-level. One only needs one detector in a room to detect whether a tag is inside or outside it. It is easy to create gates or portals to detect when tags are being moved in and out of selected areas. These portals are not small like those required for e.g. bar-code read-out in the check-out counter of a supermarket, but can be the same size as a corridor. Combined with the relatively low cost of the detector, this means that the gates/portals can be created naturally in the corridors of an institution.

In corridors or larger rooms, more detectors can be installed for a more accurate positioning (more accurate than on just room-level.) For special applications that need a very high precision, Sonitor offers the Sonitor 3DPos system, which has an accuracy of up to ± 2cm in x-, y-, and z-

coordinates. This is done by using groups of 4 to 8 detectors. Each group covers an area of 5 x 5 meters. [21], [24].

4.4.1.2 Sonitor's ultrasound tag

The tags are small, battery driven chips that emits a unique ID-number via an ultrasonic link. The tags can be placed on equipment, persons, dossiers, etc. that are to be positioned, found or secured. The emitting of the ID-number is initiated either by motion, with regular intervals, when an alarm button is pressed, or when somebody tries to remove a tag from where it is mounted. The tags come in many different variants, and can be fitted to what is needed. As an example, there are many different tags that can be used by persons and many different for equipment. [21]

The evolution of tags is however rapidly evolving, and Sonitor is currently working on developing smaller, lighter and better tags.

4.5 Positioning with RFID

An RFID system typically includes the following components:

- A tag or label that is embedded with a single chip computer, and an antenna. The antenna can be printed on the tag with carbon-based inks. The tag is an extension of the bar code labels one can see in stores today, but with more intelligence.
- A radio (much like a wireless LAN radio) that communicates with the tag. 'Passive' tags, the type of tags commonly used in retail supply chain systems, pick up enough energy from the radio to operate and to communicate back to the radio. "Active" tags have an embedded battery and offer the advantage of longer-range communications. (See below for more on active and passive tags)

Various types of tags and labels are available for use in different environmental conditions. Radios, from now on referred to as 'interrogators', can be either fixed-positioned or portable, just like bar code scanners.
An RFID system's 'read range' - the distance a tag must be from the interrogator in order to read the information stored on its computer chip - varies from a few centimetres to tens of meters, depending on frequency used, whether a tag is active or passive, and where the antenna is relative to the interrogator.

RFID tags are categorized as either active or passive.
Active RFID tags are powered by an internal battery and are typically read/write, i.e., tag data can be rewritten and/or modified. The battery-supplied power of an active tag generally gives it a longer read range. The trade off is greater size, greater cost, and a limited operational life, which may yield a maximum of 10 years, depending upon operating temperatures and battery type.
Passive RFID tags operate without a separate external power source and obtain operating power generated from the interrogator. Passive tags are consequently much lighter than active tags, less expensive, and offer a virtually unlimited operational lifetime. The trade off is that they have shorter read ranges than active tags and require a higher-powered reader. Read-only tags are typically passive.

The significant advantage of all types of RFID systems is the non-contact, non-line-of-sight nature of the technology. In addition, the tag can be programmed to hold information such as an item's serial number. Some RFID systems allow companies to write information to the tag and store it there; the RFID tag then essentially acts as a portable, dynamic database. Other systems allow the information contained on the tag to be edited, added to or locked. [25], [26]

In a hospital environment, the position of an object can be obtained when objects carrying an RFID tag pass by gates with Access Control (the interrogators).
To be used by the SAPT, this position can be communicated from the Access Control system to the Location Server.

4.6 Summary of position techniques

This section will give a brief summary of the advantages and disadvantages of the different technologies outlined in this chapter. Only techniques used for micro positioning is considered.

The advantages and disadvantages with the different tags will not be evaluated. This is because the evolution of new and better tags is very rapid, and new prototypes are presented quite often. Also, conversations with some of the developers of these technologies have revealed that they are willing to build tags that will suit the need at a hospital.
The accuracy of the different technologies is not considered either. This is because their precision can be increased by adding more infrastructure. For instance, by adding more WLAN access points, positioning with Ekahau becomes more accurate.

An evaluation of which positioning technique that is best suited for the System for Alarm, Positioning and Tracking is given in chapter 9: 'Implementation design'.

Advantages	Disadvantages
All equipment connected to WLAN is automatically included in the system	Extra cost for infrastructure of RadioEyes
No extra cabling beyond WLAN cables	All RadioEyes needs to be calibrated. If they are moved, they need to be calibrated again
	Requirements to accuracy involves a more dense placing of RadioEyes
	A PC/PDA/etc. that does not have a separate tag will not be able to be positioned if the power is switched off
	Since radio waves are used, signals from objects that are outside a room can be picked up, and the object can be registered to be on the inside
	Possible to eavesdrop outside building

Table 4–1: WLAN positioning with Radionor

Advantages	Disadvantages
All equipment connected to WLAN can easily be included in the system	Requires that special software is installed in each client
No extra physical infrastructure beyond the WLAN access points	Calibration of maps is necessary at installation of the system, and each time changes occur in the WLAN or in the buildings
	Requirements to accuracy involves a more dense placing of WLAN access points
	A PC/PDA/etc. with it's power off is not possible to position
	Since radio waves are used, objects that are outside a room can e picked up, and the object can be registered to be on the inside
	Possible to eavesdrop outside building

Table 4–2: WLAN positioning with Ekahau

Advantages	Disadvantages
No extra cabling beyond WLAN cables	All clients requires to carry a special tag
Limited by walls, therefore easy to determine on room-level	Extra cost for installing the infrastructure of detectors
Behaves like sound-waves, and does not require free sight	Requirements to accuracy involves a more dense placing of detectors
Not possible to eavesdrop outside building; high security	No positioning/alarm is possible if the tag is in a room without detector

Table 4–3: Ultrasound positioning with Sonitor

Advantages	Disadvantages
Cheap tags	All clients requires to carry a special tag
	Extra cost for installing the infrastructure of interrogators
	The position can only be obtained when passing the interrogators, i.e. not very accurate

Table 4–4: Positioning with RFID

5 Surrounding systems

This chapter will study the systems surrounding the SAPT (see figure 3–1). Special attention will be paid to the interfaces that these systems provide. This is to be able to integrate them in the overall solution. Also problems that need to be solved will be outlined.

In this thesis, the systems described here are not given further attention regarding implementation. It shall be assumed that they exist, or will exist in the future.
For example: if the SAPT needs to send a message to a certain group of people, it will just send the message to the Message Service, and the Message Service makes sure that the right person gets it.

5.1 Map System

The Map System is provided by Bravida Geomatikk AS. Maps will be available in different scales, ranging from town maps, street maps, building maps, to maps showing different floors. The Map System will present maps in the scale that users want for a graphic display of the position of objects. See figure 5–1 for example of an outdoor map.

Figure 5–1: Outdoor map showing one OI

5.1.1 Interface

The Map System can be accessed by a call to a servlet [27], which then generates the map. In the call one include the position (latitude and longitude) for the object one wants to view on the map. These longitudes and latitudes are given as decimal numbers after the formula: coordinate = degrees + minutes/60 + seconds/3600. Other parameters that can be included in the call are the scale of the map, the map's height and width, etc.
The request sent will typically be on the form:
'`<URL of the map servlet>?CX=10.438&CY=63.422&IW=550&IH=550&SC=500`'

In this example, the coordinates for an object is given, along with the desired width, height and scale of the map image.
It is possible to zoom in and out of the map that is returned as a result of the request. It is also possible to change the centre of it. [28]

5.1.2 Problems

Maps of all buildings with all floors at the hospital will have to be collected, and inserted into the Map System.
Currently, the Map System is adjusted to show maps only in the x-, y-plane (see figure 5–1). It therefore needs to be adjusted to handle z-coordinates as well, or at least a floor-coordinate (not as accurate).

The scale is not a problem. If one just has one floor in a building, the current Map System can show as precise images as needed (or as precise as the map available is).

5.2 Object Database

The Object Database (also sometimes referred to as 'the OI Database') contains information about all objects that can be positioned by the SAPT. This information could for instance be: object identification, status (out of order, free, taken, etc.), technique used for positioning this object, how often the object should be positioned, last location, log showing earlier locations, references (links) to documentation or detailed information about the object, and so on. The database can store locations of objects with x-, y- and z-coordinates. It is also possible to define objects to be an area, for instance 'the cancer department', and then specify the polygon that describes this area.

The Object Database described here is a database made at Bravida Geomatikk AS. It is run on a normal Oracle server.

5.2.1 Interface

The Object Database can be accessed through a proprietary Java API. This API offers a Java interface that makes it possible to work towards the Object Database, and communicates with it by calling procedures stored at the database. The Java API uses JDBC to invoke the stored procedures.
Many different calls can be made to insert or fetch information from the database, and if more calls are needed, this can easily be added. Because of the amount of different calls, they will not be listed here. [29]

5.2.2 Problems

The Object Database was not made for the SAPT specially. Therefore, some of the fields needed in this particular system might not exist in the present database.

5.3 Location Server

The Location Server to be used in the system is to be provided by Gintel AS. Currently however, only a prototype-version of the Location Server is available. It has been made by Telenor FoU.

The Location Server is the subsystem that collects new positions from objects and transfers them to the SAPT. The collection of positions (geographic coordinates) can be done with a range of different methods, as discussed in chapter 4. The Location Server is designed in such a way that it will be extendable when new methods for positioning from different vendors reach the market. A logical view of the server is shown in figure 5–2. [21]

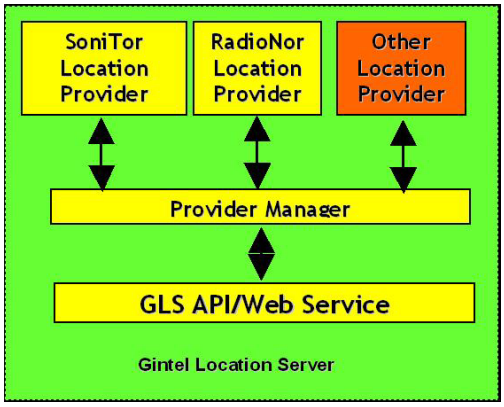


Figure 5–2: Logical view of Gintel Location Server

5.3.1 Interface

The Location Server provides a web service/JMS interface for applications. The main functionality is a set of requests/responses, and a framework where one can subscribe to location related incidents.

The Location Server currently provides 5 different messages: `FIND`, `TRACK`, `NOTIFY`, `NOTIFY_WHEN_MOVED` and `DELETE`. These messages can be sent from a client to the Location Server. The client then waits for an answer, and will receive one or more replies depending on the message it sent. An explanation of the different messages is given below. [30]

`FIND:`
This message is used if a client wants to know where an object is 'just now'.
The client sends a request to the server, asking to find the object. He receives only one reply, containing the current location for the object.

`TRACK:`
This message is used if a client wants to monitor an object (or objects), i.e. wants to know where an object is at any time.
The client sends a request to the server, asking to start tracking the object. He then receives a series of replies at regular intervals. The replies contain new positions for the object. The 'tracking interval' (how often the server shall send a new position to the client) can be set by the user when a `TRACK` messages is initiated.

`NOTIFY_WHEN_MOVED:`
This message is used if a client wants to receive a notification when an object is moved.
The client sends a request to the server, and will then only receive replies if the object is moved. It is possible to include a distance in the request, which indicates that one only wants to receive a reply if the object is moved more than this distance. This can be a very desirable feature, as it is for instance not very interesting to be notified if a bed has been moved 5 cm. One could then say that one only wants to receive replies if the bed is moved 3 meters or more.

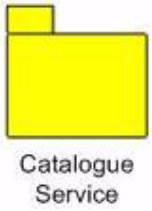
NOTIFY:
This message can be used if a client wants to be notified when an object is, for instance, moved in to a prohibited area (example: a visitor/thief walks into the control room), or to get alarm if an object is moved out of the area it belongs to (example: somebody tries to steal a TV).
The client sends a request to the server, and will then only receive replies when/if the object is moved to prohibited areas. In the request one includes a point (x- and y-coordinates), which will be the centre of the area. At the moment it is only possible to specify if the area should be a circle or a square, but this will later be extended to cover any geometric shape. In addition, one specifies if one wants a response when the object is moved *out* of the area, or when the object is moved *in* to the area.

DELETE:
This message terminates the connection between the client and the Location Server.

5.3.2 Problems

The current Location Server is just a prototype, and will have to be developed further to be a more mature product.
Priority on messages needs to be implemented. This so that important location updates, for instance from alarms, gets to the right persons very fast. Other aspects that must be emphasized are delay on messages, and that the server needs to be able to handle very many requests at the same time.

5.4 Catalogue Service



The Catalogue Service is an important component to support security. The need for mechanisms to handle requests concerning authentication and authorization requires an infrastructure capable of handling identities and resources. The Catalogue Service is a basis to administrate users, user accounts, access rights to software, services, equipment, etc., at the hospital.
The Catalogue Service shall be the central tool for coordination and administration of all users and the user profiles for these, with special focus on user- and group-profiles. It shall also administrate end-user equipment/clients and their configurations. In addition the Catalogue Service offers a search-function that makes it possible to search for information regarding persons and equipment at the hospital. This can be such things as name and phone number for persons, and user manuals for equipment. [31]

5.4.1 Interface

The interface is not specified, as the Catalogue Service is not implemented yet.

5.4.2 Problems

The Object Database (see section 5.2) and the Catalogue Service will contain some identical information. This will typically be information relating to equipment and staff at the hospital, and the SAPT will need to handle this.
Things that need to be determined are: how common data about the objects shall be stored and potentially replicated, and how to update common data for the objects.

The Object Database will also provide links to documentation for objects. The actual documentation however, will typically be stored in the Catalogue Service. A problem is how to keep these links up to date.
In addition, the Catalogue Service contains access control data for all employees at the hospital, such as usernames, passwords and user profiles. When a user tries to log in to the SAPT with a username and password, the SAPT will need to access the Catalogue Service to authenticate the user. The Catalogue Service then validates the username and password, and returns a response with the right access level and user profile for this user.

5.5 Message Service

The Message Service is a service that will handle all signalling between instances at the hospital. It is a common interface that routes and forwards alarms and messages. The Message Service needs to be able to receive messages from the SAPT, and to forward them to whoever the receiver is. It shall also be possible to send messages to the SAPT from other systems by means of the service. The Message Service (and systems behind) makes sure that the right persons get the different messages/alarms. [7]

5.5.1 Interface

The interface is not specified, as the Message Service is not implemented yet.

5.5.2 Problems

How to specify the interface between the Message Service and the SAPT is a problem that will have to be solved.
The SAPT needs to be able to send alarms. The desired behaviour is that it just sends an alarm message to the Message Service, and says which object the alarm is for. The Message Service (and systems behind) has to make sure that the proper person(s) receives the alarm.
The SAPT also needs to be able to receive an alarm from other systems than the Location Server via the Message Service. This can be alarms from the fire alarm system, from the access control system, etc.

6 Objective and methodology

This chapter will give a brief revisit to the purpose of the task, before the methodology used to design the system is described.

6.1 A revisit to the purpose of the task

So far this thesis has given an introduction to a new problem area. It has described the current routines at St. Olav's Hospital in Trondheim, Norway, and pointed out that in certain areas, the routines are ineffective. The thesis then presented the structure for a new system, and gave examples of application areas for such a system.
Through this, the reader hopefully understood the need for the SAPT at the hospital. As pointed out, the system can also be used in other fields as well, such as universities and other big buildings. That the system is not locked to just one application increases the value of the work done in this thesis.
An introduction to positioning was also given, along with different techniques that can be used to obtain positions of an object. The systems that the SAPT will interface were then given a brief explanation.

The contribution of this thesis is to make use of the techniques described in chapter 4, and the systems described in chapter 5, to design a system as depicted in chapter 3. No formal design or concrete requirements specification exist, so this will have to be made. A major problem that has to be solved is that there will be many different systems that need to communicate with each other in an efficient way.

The SAPT will add values for users in the areas concerning tracking objects, finding objects and getting alarms.

6.2 Methodology

The methodology employed in this thesis is based on [32]. The approach uses a step-wise method, that consists of specifying the requirements applying to the system, then designing the system, and then implementing it. This is shown in figure 6–1.

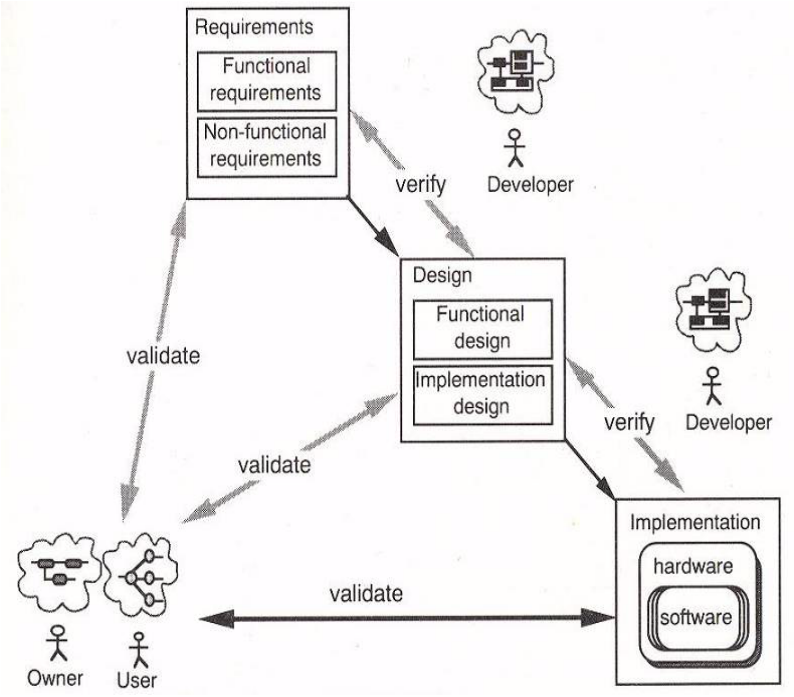


Figure 6–1: Step-wise methodology for systems engineering

According to the recommendations in [32] there has been made a clear separation between WHY, WHAT and HOW when making the system. The requirements specification will tell us WHY the system is needed. The design tells WHAT the system will do. The implementation tells HOW the functions are to be realised in an actual implementation for a concrete system. The implementation is concerned with the concrete system, while the design might be used in several different systems that are to provide the same functionality.

The idea is to develop the requirement specification first, then to use the functional requirements as a basis for a functional design, then to map the functional design and the non-functional requirements into an implementation design, and finally to create a concrete implementation of the system.

As shown in figure 6–1, the owner and the user are important in the process. In order to obtain a good product, they should validate all the steps. In this study there will be no actual user to talk to. The role of owner can however be filled by the mentor of this thesis, and he can then validate the different steps of the process.

7 Requirements specification

Definition of requirements:
The requirements for a software system are a complete description of the external behaviour of the system that is going to be implemented. [33]

The first step in a system engineering process is to find the requirements that apply to a system. A requirements specification will by no means dictate how to implement a system, just what the system is supposed to do. It captures the intended behaviour of the system.

This chapter will first give a brief introduction to who will be the users of the system. Next, it will take a look at the concept of actors and roles. The functional and non-functional requirements applying to the system will then be presented, before this chapter ends by describing suggestions for the user interface of the system.

7.1 Who will be using the system?

Users of the SAPT will mainly be employees at the hospital. Patients and visitors will also be affected by the system, they can trigger alarms for instance, but they will never access the system through a terminal.
The system will have to be designed to suit the technical level of the users. Since one can not expect that all employees have very detailed computer skills, the system needs to be intuitive, and not too hard to operate. A long training period should not be needed to start using it.

7.2 Actors and roles

This section will give an introduction to the concept of actors and roles. After that, the different roles that will be present in the SAPT is described.

According to [34], *'An **actor** is a user or external system with which a system being modelled interacts'*. In the SAPT, an actor can for instance be Per Hansen, or a special dialysis machine. These actors are said to play ***roles*** in the system studied. A role represents a grouping of

functionality that is associated with different types of tasks. Each actor can play more that one role. For example: Per Hansen can both play the role as a visitor to the hospital, and as a doctor at the same hospital. There are many advantages in introducing the actor/role concept. One of these is to avoid having tasks locked directly to individuals. [34], [32]

It has been identified 9 roles that are relevant to the SAPT. These are:

- Equipment
- Visitor
- Patient
- Nurse
- Doctor
- Hospital orderly
- Guard
- Safety responsible
- Administrator

It has been recognized that some of these roles will behave equally in the system, and can therefore be combined to one role. The resulting 4 different roles will be:

Name:	Roles included:
Role 1:	Equipment/Visitor/Patient
Role 2:	Nurse/Doctor
Role 3:	Hospital orderly/Guard
Role 4:	Safety responsible/Administrator

Table 7–1: The different roles in the system

All of these roles will have different capabilities in the system.

Access rights and other security aspects are not considered. If they where, there would be more than just 4 different roles. Maybe as many as the 9 original ones. The different roles are merged solely on the basis of their functional behaviour in the system.

7.3 Functional requirements

It is normal to divide the requirements to a system in functional requirements and non-functional requirements. According to [32], '*functional requirements are concerned with the system services, i.e. the behaviour as the user will see it. They are the primary inputs to the functional design. The non-functional requirements, on the other hand, are constraints on the implementation of the system. They are inputs to the implementation design.*'

This section will study the functional requirements applying to the system, while section 7.4 will describe the non-functional requirements.

7.3.1 Functional requirements relating to actors

This section will describe which functions that actors shall be able to perform by means of the SAPT.

It is found and described 10 functional requirements that the system has to provide to actors. They are listed in table 7–2.

Requirement:	Description:
FA 1:	An actor shall be able to log in to the system
FA 2:	An actor shall be able to manage objects
FA 3:	An actor shall be able to update object parameters
FA 4:	An actor shall be able to be positioned
FA 5:	An actor shall be able to search for objects and show position of these
FA 6:	An actor shall be able to get information on objects
FA 7:	An actor shall be able to trigger alarm
FA 8:	An actor shall be able to monitor objects
FA 9:	An actor shall be able to receive alarm
FA 10:	An actor shall be able to log out of the system

Table 7–2: Functional requirements relating to actors

Following are these requirements described in detail. The notation of text based filled use cases is used for the description. See appendix A: ’Text based filled use cases -notation’ for a specification of this notation.
A thing to note is that not all actors are allowed to perform all use cases. Which actors that can perform which use cases depend on their current role. This will be outlined in section 7.3.2: ’Roles related to use cases’.

Terminology used in this chapter:

- *The system:* The overall system that this thesis is describing, i.e. the SAPT.
- *Terminal:* Any device used for accessing the system. Typically: PC, PDA, laptop, etc.
- *Actor:* Actors play roles in the system. See section 7.2.
- *Object:* An object is the same as an OI. An OI can have any of the 9 roles listed in section 7.2, for instance ‘doctor’ or ‘equipment’.

7.3.1.1 FA 1: Log in

The functional requirement concerning logging in is described in table 7–3.

Use case name	Log in
Iteration	Filled
Summary	An actor becomes logged in to the system
Basic eventpath	1. The actor presents his username and password to the system 2. The actor is successfully logged in to the system with the privileges applying to his role
Alternative paths	-
Exceptional paths	In step 2, the actor is not allowed to log in to the system
Trigger	An actor wants to log in to the system so that he can start using it
Pre condition	The actor has to be connected to a terminal
Post condition	The actor has been logged in to the system
Author	Solrun Pedersen
Date	20/3-2004

Table 7–3: Text based use case for functional requirement FA 1

7.3.1.2 FA 2: Manage objects

The functional requirement concerning managing objects is described in table 7–4.
In this context, to manage objects means to add an object to the system, or to delete an object from the system.

Use case name	Manage objects
Iteration	Filled
Summary	An actor adds or deletes an object in the system
Basic eventpath	1. The actor has a new object he wants to add to the system 2. The actor tells the system he wants to add a new object 3. The system lets the actor submit information relating to the new object 4. The new information is stored by the system
Alternative paths	1. The actor selects an object he wants to delete 2. The actor tells the system he wants to delete this object 3. The system deletes this object
Exceptional paths	The system could not add/delete the object
Trigger	An actor wants to or add or delete an object
Pre condition	The actor has to be connected to a terminal, and use case *Log in* must have been executed

Table 7–4: Text based use case for functional requirement FA 2

Post condition	An object has been deleted from the system, or a new object has been created
Author	Solrun Pedersen
Date	20/3-2004

Table 7–4: Text based use case for functional requirement FA 2

7.3.1.3 FA 3: Update object parameters

The functional requirement concerning managing objects is described in table 7–5.
In this context, an 'update object parameters' call is a collective term used for all calls involving object parameters that do not involve adding or deleting objects.

Examples of calls can be:

- Start/stop tracking for an object
- Set/unset alarm for an object
- Change the zone for an object
- Free an object from a zone, so that it can be moved between zones (wards) without triggering alarms
- Update attributes belonging to an object

Use case name	Update object parameters
Iteration	Filled
Summary	An actor updates parameters relating to a specific object
Basic eventpath	1. The actor selects the object he wants to update 2. The actor sends the update-parameters to the system 3. The system updates the correct object with the correct information
Alternative paths	-
Exceptional paths	In step 3, the system could not update the object
Trigger	An actor wants to update some or all of the parameters of an object
Pre condition	The actor has to be connected to a terminal, and use case *Log in* must have been executed. In addition, use case *Manage objects* must have been performed on the object that is to be updated (to add it to the system)
Post condition	The object is updated with new information
Author	Solrun Pedersen
Date	20/3-2004

Table 7–5: Text based use case for functional requirement FA 3

7.3.1.4 FA 4: Be positioned

The functional requirement concerning being positioned is described in table 7–6.

Use case name	Be positioned
Iteration	Filled
Summary	The position of an actor gets recorded in the system
Basic eventpath	1. The system is set up to regularly track the position of this actor 2. At intervals, the system fetches the position of the actor 2. The actor's position is recorded in the system
Alternative paths	Path 1: 1.1. Another actor in the system wants to know the position of this actor 1.2. The system is not set up to track the position of the actor 1.3. The system sends a message to this actor, asking for his location 1.4. The actor's position is recorded in the system Path 2: 2.1. An alarm is triggered for this actor. This can be if the actor has moved outside a legal area, has pressed an alarm button on a tag, etc. 2.2. The current location of the actor is immediately sent to the system 2.3. The actor's position is recorded in the system
Exceptional paths	The position of the actor could not be obtained. The reason for this might be: the actor has moved out of the area covered by the system, the actor's device used for positioning is out of order, etc.
Trigger	Either the actor has triggered an alarm, or someone (the system or another actor) want to know the position of this actor
Pre condition	The object to be positioned must possess a device (tag) that can be positioned, and use case *Manage objects* must have been performed on it
Post condition	The position of the actor is recorded in the system
Author	Solrun Pedersen
Date	20/3-2004

Table 7–6: Text based use case for functional requirement FA 4

7.3.1.5 FA 5: Search for objects and show position

The functional requirement concerning searching for objects, and then showing the position of these is described in table 7–7.

Use case name	Search for objects and show position
Iteration	Filled
Summary	An actor searches for objects matching certain parameters, and then shows the position of the objects on a map
Basic eventpath	1. The actor accesses a search page, and specifies a search based on different parameters (for instance: type, ward, room, closest to my location, etc.) 2. The system returns a result of the search as a list of matching objects 3. The actor selects the objects that he wants to view location of 4. The locations of the objects are shown on a map
Alternative paths	In some cases the actor only wishes to show the location of one single object straight away. This might for instance be if a fire alarm goes off. He can then just input the unique ID of the object as a parameter to the system, which then shows the position directly
Exceptional paths	The location of the object can not be obtained, and the map can not be shown
Trigger	An actor wants find one or more objects that matches certain criteria
Pre condition	The actor has to be connected to a terminal, and use case *Log in* must have been executed. In addition, use cases *Manage objects* and *Be positioned* must have been performed on the objects that are being searched for
Post condition	The position of the selected objects are shown on the screen of the actor's terminal
Author	Solrun Pedersen
Date	20/3-2004

Table 7–7: Text based use case for functional requirement FA 5

The reason for not dividing this use case in two different use cases; i.e. one use case for 'search for objects' and one for 'show position' is the following: To be able to search for objects that are close to an actor, the parameters that the actor inputs needs to contain his current location. The search procedure then has to get the locations for all objects matching the parameters, to be able to show the list. Most of the job that a separate use case 'show position' would have performed is then done, as the locations for all objects is already obtained. It therefore seemed more time saving to merge the two situations in one use case.

7.3.1.6 FA 6: Get information on objects

The functional requirement concerning getting information on objects is described in table 7–8.

Use case name	Get information on objects
Iteration	Filled
Summary	An actor gets information on an object
Basic eventpath	1. After the search in use case *Search for objects and show position*, the actor wants to get some more information on one of the objects that was given as a result 2. The actor either selects an object from the list, or clicks directly on the desired object on the map 3. The system then shows some basic information about the object, such as status, current room, a short description, etc. A list with links to other information that exists on the object, such as user manuals, documentation, statistics, etc., will also be shown 4. The actor selects one of the links from the list 5. The desired information is presented to the actor
Alternative paths	-
Exceptional paths	Information on the selected object is not entered in to the system, so the actor can not obtain the desired information
Trigger	The actor wants to get information on an object
Pre condition	The actor has to be connected to a terminal, and use case *Search for objects and show position* must have been executed
Post condition	The actor has received information on the object
Author	Solrun Pedersen
Date	20/3-2004

Table 7–8: Text based use case for functional requirement FA 6

7.3.1.7 FA 7: Trigger alarm

The functional requirement concerning to trigger alarm on objects is described in table 7–9.

Use case name	Trigger alarm
Iteration	Filled
Summary	An actor in the system triggers an alarm
Basic eventpath	1. A new position for an actor is received by the system 2. The system detects an alarm for the actor with this new position. This can for instance be because the actor has moved in to an illegal area
Alternative paths	1. An actor that possesses a tag with an alarm button, presses the button 2. An alarm message is sent from the tag to the system 3. The system detects the alarm

Table 7–9: Text based use case for functional requirement FA 7

Exceptional paths	-
Trigger	The system receives a new message from an object
Pre condition	The object to be positioned must possess a device (tag), and use case *Manage objects* must have been performed on it
Post condition	An alarm is triggered
Author	Solrun Pedersen
Date	20/3-2004

Table 7–9: Text based use case for functional requirement FA 7

7.3.1.8 FA 8: Monitor objects

The functional requirement concerning monitoring objects is described in table 7–10.

Use case name	Monitor objects
Iteration	Filled
Summary	An actor monitors one or more objects from a terminal
Basic eventpath	1. The actor selects the objects he wants to monitor 2. The system shows the desired objects on a map. The actor can see the objects move across the map
Alternative paths	-
Exceptional paths	The location of some of the objects can not be obtained
Trigger	An actor wants to monitor one or more objects
Pre condition	The actor has to be connected to a terminal, and use case *Log in* must have been executed. In addition, use cases *Manage objects* and *Be positioned* must have been performed on the objects that are being monitored
Post condition	A map with moveable objects is shown on the screen of the actor's terminal
Author	Solrun Pedersen
Date	20/3-2004

Table 7–10: Text based use case for functional requirement FA 8

7.3.1.9 FA 9: Receive alarm

The functional requirement concerning receiving alarm is described in table 7–11.

Use case name	Receive alarm
Iteration	Filled
Summary	An actor receives an alarm
Basic eventpath	1. An alarm is detected by the system 2. The system finds out which actors are supposed to receive the alarm. This particular actor is one of them 3. The system sends a message to the actor, which is shown on a screen on his terminal. The message contains information about which object that did trigger the alarm, and a map showing where the object is 4. The actor acknowledges that he has seen the alarm by pressing a button 5. The alarm message is removed from the screen
Alternative paths	-
Exceptional paths	If the system is not up to date, the alarm message is sent to the wrong actors
Trigger	An object triggers an alarm
Pre condition	The actor has to be connected to a terminal, and use case *Log in* must have been executed. In addition, use cases *Manage objects* and *Trigger alarm* must have been performed on an object
Post condition	An alarm-notification is shown on the screen of an actor's terminal
Author	Solrun Pedersen
Date	20/3-2004

Table 7–11: Text based use case for functional requirement FA 9

7.3.1.10 FA 10: Log out

The functional requirement concerning logging out is described in table 7–12.

Use case name	Log out
Iteration	Filled
Summary	An actor is logged out of the system
Basic eventpath	1. The actor tells the system that he does not want to be logged in on the system any more 2. The actor is successfully logged out
Alternative paths	If the actor has been passive for some defined time, he is automatically logged out
Exceptional paths	The system may malfunction, and the actor gets disconnected without first sending a log out request
Trigger	The actor wants to stop using the system

Table 7–12: Text based use case for functional requirement FA 10

Pre condition	The actor has to be connected to a terminal, and use case *Log in* must have been executed
Post condition	The actor is disconnected from the system
Author	Solrun Pedersen
Date	20/3-2004

Table 7–12: Text based use case for functional requirement FA 10

7.3.2 Roles related to use cases

This section will outline which use cases relate to which roles, as not all roles should be able to perform all use cases. For instance; a visitor to the hospital should not be able to delete objects from the system, but this will typically be something an administrator is allowed to do. Below, this is shown by using use case diagrams along with a short description.

7.3.2.1 Role 1: Equipment/Visitor/Patient

An actor with role type 1 will only interfere with the system in two ways: he can be positioned, and he can trigger alarm. This is shown in figure 7–1.

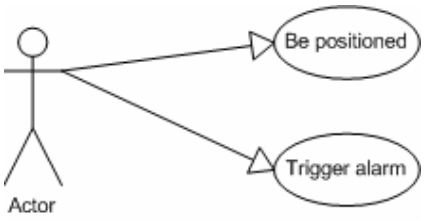


Figure 7–1: Use case for role type 1

7.3.2.2 Role 2: Nurse/Doctor

An actor with role type 2 should be able to perform the use cases shown in figure 7–2.
The actor shall be able to log in to the system, and use it to find persons and equipment. He shall be able to update object parameters, for instance to set the status for an equipment to ‘occupied’, to update the room number a patient belongs to, etc., and to get information relating to objects. This actor shall also be able to trigger alarm. This can for instance be an assault alarm if a mental patient is being violent. This actor therefore needs to be able to be positioned too, so one can find the location of the actor that triggered the alarm.

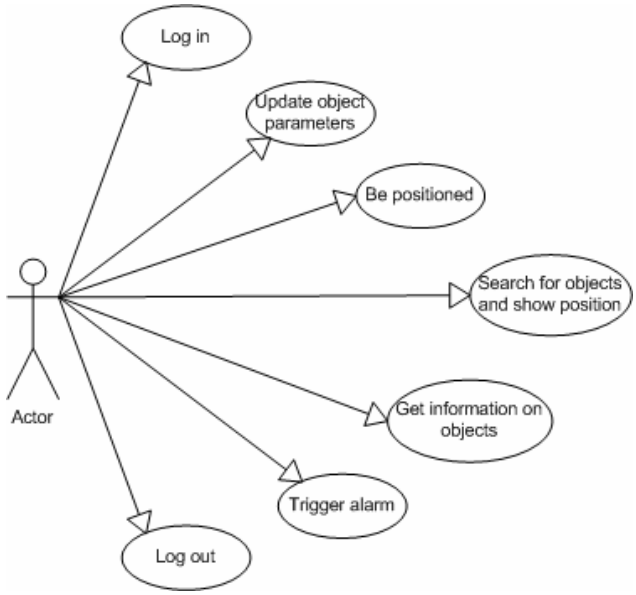


Figure 7–2: Use case for role type 2

7.3.2.3 Role 3: Hospital orderly/Guard

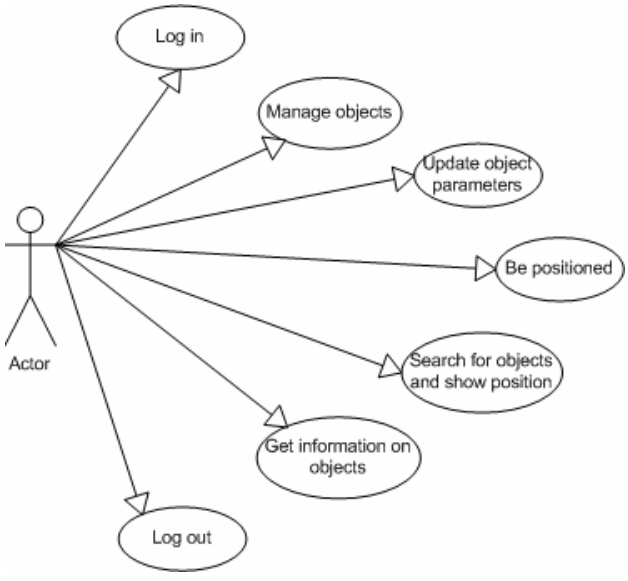


Figure 7–3: Use case for role type 3

An actor with role type 3 should be able to perform the use cases shown in figure 7–3.
A hospital orderly or a guard will typically move equipment, for instance beds, around the hospital. He will therefore be dependent on the function ‘update object parameters’. This is

because he needs to be able to deactivate alarms for equipment so they can be transferred to another location without triggering alarm, and other similar operations. When the equipment has reached its new location, a new alarm can be activated for this object.
This actor will typically be the one that receives new equipment arriving at the hospital, so it will be natural that he is able to add and delete objects from the system (use case 'manage objects'). He will also sometimes need to find objects that he is to transfer, and maybe get some information regarding them. Other actors in the system might need to find a hospital orderly, and therefore role type 3 needs to be able to be positioned.

7.3.2.4 Role 4: Safety responsible/Administrator

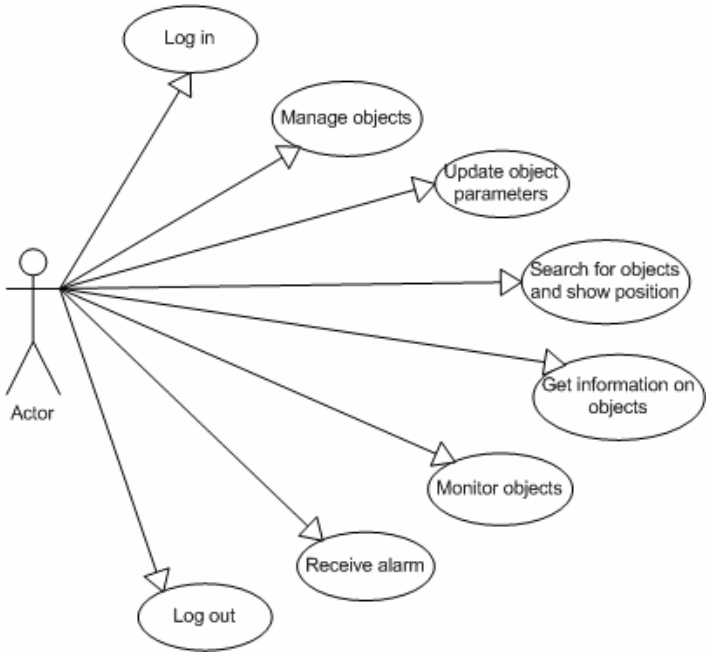


Figure 7–4: Use case for role type 4

An actor with role type 4 should be able to perform the use cases shown in figure 7–4.
This actor will typically sit in a separate room and monitor the system. He needs to be able to manage objects, and to update parameters for these. An actor with this role will not have a particular need to be positioned, as he will mostly be sitting on the same spot.
A thing to note is that by means of the SAPT, this actor will be the only one that can receive alarm. Other actors will of course receive alarms too. For instance, if a patient presses the alarm button on his alarm-tag, an alarm-message should be sent to the nurse that is in charge of this patient right now. This will however *NOT* be handled by the SAPT. It will only detect alarms, find out which object the alarm is for, and use the Message Service to send the alarm. Other systems will then determine which physical person(s) that shall receive the alarm.
An actor in the system with role 4 should receive all alarms triggered at the hospital. Some of these alarms will be detected by the SAPT itself. Other alarms will be detected elsewhere in the hospital, and an alarm message will be sent to the SAPT via the Message Service.

7.3.2.5 Roles and access mechanisms

A note is needed on the mechanisms that the different roles shall use to access the system.

Actors in the system with role 1 will not access the system through a terminal. These will not have a password or username in the system either.

It has been decided that actors logged in to the system with roles 2 and 3 shall access the system through web pages accessible from the Internet. The system shall then be available for these actors from any terminal with an Internet connection.

A normal web-based system operates in a request-response way. This means that an actor can only receive a response from a web server if he has sent a request first. This is just fine for an actor with role 2 or 3. A web based system can however not send a message to an actor without receiving a request first. An alarm message can for instance not be sent directly to an actor. Actors in the system with role 4 need to be able to receive alarms, so a web-based solution is clearly not suitable for these. A Java application on the other hand can receive independent messages from other systems, without having sent a request first. It has therefore been decided that actors with role 4 shall access the system through a Java application.

7.3.3 Functional requirements to the SAPT

This section will specify the functional requirements applying to the SAPT. Only requirements that are not directly related to those described in section 7.3.1 are included. For instance, for an actor to be able to log in to the system, the SAPT has to provide mechanisms for logging in, but this is not repeated in this section.
All text in quotation marks (‘’) refers to section 7.3.1.
The list of requirements is given in table 7–13.

Requirement:	Description:
FS 1:	The system must be able to handle ‘manage objects’ request from the Catalogue Service *Comment: If an actor adds or deletes objects in the Catalogue Service, this information also needs to be stored in the Object Database*
FS 2:	The system must be able to handle ‘update object parameters’ request from the Catalogue Service *Comment: If an actor updates parameters in the Catalogue Service, this information also needs to be updated in the Object Database*
FS 3:	The system must be able to handle ‘search for objects and show position’ request from the Message Service *Comment: Other systems at the hospital might want to make use of the ability the SAPT provides for viewing the location of an object on a map. This can for instance be to view the location of a triggered fire alarm*
FS 4:	The system must be able to handle alarm messages from the Message Service *Comment: This is because an actor logged in with role 4 shall receive all alarm messages in the hospital, including those not triggered by the SAPT*
FS 5:	The system must be able to keep logs over earlier positions for objects

Table 7–13: Functional requirements to the SAPT

For a graphical interpretation of the functional requirements specified in chapter 7.3.1: 'Functional requirements relating to actors' and chapter 7.3.3: 'Functional requirements to the SAPT', please consult appendix B: 'Graphical interpretation of use cases'. This is recommended to be able to understand the system in more detail. The figures in the appendix also give a good explanation of how the different components need to communicate to fulfil the tasks.

7.3.4 Functional requirements relating to a search

As described in FA 5, an actor shall be able to perform a search for objects. The search shall be performed based on parameters that the actor can input to the system via his terminal.

Table 7–14 summarizes the requirements that apply to the search parameters.

Requirement:	Description:
FSE 1:	An actor shall be able to search for an object based on its *unique identification number*. This search will return only one OI
FSE 2:	An actor shall be able to search for a person based on the person's *name*
FSE 3:	An actor shall be able to search for a *type* of object (bed/dialysis machine/doctor). This search will return all the OIs of this type
FSE 4:	An actor shall be able to narrow his search after selecting the type, by adding the parameter '*building*'. If the actor is logged in with role 4, he can choose only this parameter, to be able to monitor a whole building
FSE 5:	An actor shall be able to narrow his search after selecting the type, by adding the parameter '*ward*'. If the actor is logged in with role 4, he can choose only this parameter, to be able to monitor a whole ward
FSE 6:	An actor shall be able to narrow his search after selecting the type, by adding the parameter '*floor*'. If the actor is logged in with role 4, he can choose only this parameter, to be able to monitor a whole floor
FSE 7:	An actor shall be able to narrow his search after choosing the type, by adding the parameter '*room*'. If the actor is logged in with role 4, he can choose only this parameter, to be able to monitor a whole room
FSE 8:	An actor shall be able to narrow his search after selecting the type, by adding the parameter '*status*' (for instance 'free'/'occupied'/etc.). This parameter can be combined with any of the parameters in **FS 4 - FS 7**

Table 7–14: Functional requirements relating to a search

If name or unique identification number is chosen, the other parameters will have no effect if they are chosen as well.

7.3.5 Functional requirements relating to Objects of Interest (OIs)

This section will state the functional requirements that apply to the objects that will be present in the Object Database. The requirements are shown in table 7–15.

Requirement:	Description:
FO 1:	An object shall be classified in to one of two groups; *Object OI* or *Area OI*
FO 2:	An Object OI shall be either of type *stationary* or of type *dynamic*
FO 3:	An Area OI shall be either of type *room* or of type *zone*

Table 7–15: Functional requirements relating to OI's

The reason for this particular classification is that one shall be able to set up some OIs to belong to other OIs. Specifically, an Object OI can belong to an Area OI. Example: 'bed number 12345 belongs to room A-123'. To be able to specify an area, defined with geographic coordinates, one need an Area OI.

Examples of the different types of OIs:

- *Stationary Object OI*: Will typically be a TV, the place a fire alarm is situated, etc.
- *Dynamic Object OI*: Will typically be beds, medical equipment, staff, etc.
- *Room Area OI*: Will typically be 'room-345', etc. This can be used to give the actor a textual description of the current location of an Object OI
- *Zone Area OI*: It is possible to insert any geographic area into the Object Database, for instance a square of a selected size, or any other self-defined area (polygon). As an example, a Zone Area OI can be used to define the exact boundaries for the cancer department

7.3.6 Functional requirements relating to tags

This section will state the functional requirements that will apply to the tags to be used in the system. Tags are needed by the location mechanisms.
As mentioned before, the tags and the location system is not to be designed in this thesis. It is still important to state requirements to these tags, as different tags are needed for different usages in the system.

The following sections are based on the roles applying to actors in the system.

7.3.6.1 Tags for equipment

Tags for equipment must be easy to attach, but hard to remove. One can imagine that some tags have a sort of glue-mechanism that makes it easy to stick on to certain equipment, for instance beds. For other equipment, for instance TV's, the tag can be mounted inside the equipment's chassis, to make it even harder to remove.
These tags should be equipped with a mechanical sensor to detect if somebody tries to remove them from the equipment. If this happens, an alarm should go off, both as a message to the SAPT, and as sound from the tag to attract the attention to people around. The same should happen if the equipment is moved outside the zone it belongs to.
This is summarized in table 7–16.

Requirement:	Description:
FTE 1:	Tags for equipment shall be easy to fasten, but hard to remove
FTE 2:	If somebody tries to remove a tag for equipment, it shall give alarm, both as a message to the SAPT and as sound from the tag to attract the attention of people nearby
FTE 3:	If somebody tries to remove equipment with a tag from the zone it belongs to, the tag shall give alarm, both as a message to the SAPT and as sound, to attract the attention of people nearby

Table 7–16: Functional requirements relating to tags for equipment

7.3.6.2 Tags for visitors

Tags for visitors are needed to make sure that visitors do not enter illegal areas of the hospital. In addition, if an alarm goes off for some equipment, it can be interesting to see which visitors that are close by. Usually however, visitors are only at the hospital for a short period, and most of them just wants to see their loved ones, and do not want to steal a TV.

There will be a trade-off in the functionality for visitor tags.
Firstly, the tags need to be easy to fasten and take off persons. Since it potentially will be very many visitors, the guards can not spend a long time on each of them. The tags should also be reusable, so one does not have to throw away tags after one visitor has used it.
Secondly, the tags should be difficult to remove, and preferably trigger alarm if someone tries to remove them. Otherwise, they will be of no use. If a visitor wants to go into an illegal area, he should not be able to remove his tag easily.
It is therefore a trade-off between a handy and easy-to-use tag, and that the tag should be hard to remove.
This is summarized in table 7–17.

Requirement:	Description:
FTV 1:	Tags for visitors shall be easy to fasten and remove by trained staff
FTV 2:	Tags for visitors shall be reusable
FTV 3:	Tags for visitors shall be hard to remove by the visitor. If the visitor tries to remove his tag, the tag shall give alarm, both as a message to the SAPT and as sound from the tag to attract the attention of people nearby
FTV 4:	If a visitor carrying a tag enters an illegal zone, the tag shall give alarm, both as a message to the SAPT and as sound, to attract the attention of people nearby

Table 7–17: Functional requirements relating to tags for visitors

A question to ask is ‘are tags for visitors really of any use?’ This important question has to be discussed. The hospital can use other means to limit the access to illegal areas, for instance by using doors with locks.

7.3.6.3 Tags for patients

Two different types of tags to be used for patients are needed:

- Mobile or stationary tags with a button that can be used to signal alarm. Examples are if an elderly person needs help to go to the toilet, or has fallen over and can not get up
- Tags to be attached to unstable patients, for instance mental patients, to keep track of where they are

One can discuss if all patients should be tracked, or if this should only be done on certain groups of patients (for instance mental patients). See the discussion under 'Tags for visitors'.
The tags to be used to signal alarm can either be stationary, for instance mounted to the wall in a patient's room, or mobile. If the tag is mobile, the patient can have it in a 'necklace' around his neck, or around his arm. This tag does not have to be hard to remove. When the button is pressed it gives alarm, both as a message to the SAPT, and as sound from the tag to attract the attention to people around.
The tag that shall be attached to patients to keep track of where they are can for instance be put around their hand. It should give alarm, both as a message to the SAPT, and as sound from the tag to attract the attention of people around if somebody tries to remove it. The same should happen if the patient moves outside permitted zones.
This is summarized in table 7–18.

Requirement:	Description:
FTP 1:	There shall exist alarm tags that patients can get if wanted
FTP 2:	Alarm tags shall, if pressed, give alarm. Both as a message to the SAPT, and as sound from the tag to attract the attention of people nearby
FTP 3:	There shall exist tags for patients that can be used to keep track of where a patient is
FTP 4:	Tracking-tags shall be hard to remove. If somebody tries to remove them, it should give alarm, both as a message to the SAPT, and as sound from the tag to attract the attention of people nearby
FTP 5:	If a patient carrying a tracking-tag moves outside permitted zones, the tag shall give alarm, both as a message to the SAPT and as sound, to attract the attention of people nearby

Table 7–18: Functional requirements relating to tags for patients

7.3.6.4 Tags for staff

Two different types of tags to be used for staff are needed:

- Tags to position staff, to be able to find them when needed
- Tags to signal alarm, for instance alarm if a staff is assaulted by one of the patients

Tags to position staff should preferably be integrated in the staff's wireless terminals. If WLAN positioning is used, for instance, all PDAs at the hospital are automatically included in the system, and do not need a separate tag. Another solution would be to stick a tag on to the staff's mobile telephone, or to their ID-card. This does however require that the tags are much smaller and lighter than they are today.
Tags to signal alarm could be integrated in the staff's wireless terminals, either as a 'soft-key' on the staff's mobile telephone/PDA, or as a separate tag with a button that can be pressed to

indicate alarm. This tag can be worn as a 'necklace' around the neck, or around the arm. When the button is pressed it shall give alarm, both as a message to the SAPT, and as sound from the tag to attract attention from people around.
It is a desire to have an accelerator detector in the tag that can detect if one of the staff falls to the floor. This can be used in cases where a nurse is attacked by a patient, for instance, and does not manage to press the button on his tag before the assault.
Tags for staff do not have to be hard to remove.
This is summarized in table 7–19.

Requirement:	Description:
FTS 1:	There shall exist tags for staff that can be used to keep track of them
FTS 2:	Tracking-tags shall be integrated equipment the staff already possess
FTS 3:	There shall exist alarm tags for staff
FTS 4:	If an alarm-tag detects an alarm, it shall give alarm both as a message to the SAPT, and as a sound from the tag to attract the attention of people nearby

Table 7–19: Functional requirements relating to tags for staff

7.4 Non-functional requirements

Non-functional requirements are, in contrast to functional requirements, hard to connect to a specific part of an application. They are rather properties of the application as a whole. The non-functional requirements to a system decide which implementation design one ends up with, given that the functional requirements are made clear.

This section of non-functional requirements is based on IEEE's standard 830-1993 for recommended practice for software requirements specifications [35]. From the guidelines in this document, it has been chosen to divide the requirements in four groups: interface requirements, performance requirements, technical requirements and quality requirements respectively.

Because of limited time available when working with this thesis, the non-functional requirements have not been studied in detail. The requirements given here should be considered as examples on the different types, rather than a full specification of the non-functional requirements that apply to the SAPT.

7.4.1 Interface requirements

Interface requirements specify the interaction between actors, hardware and software. In this section, requirements to the following will be stated:

- The user interface between the system and the system's users
- The communication interface between software components in the system and external systems

The interface requirements applying to the SAPT is stated in table 7–20.

Requirement:	Type:	Description:
NFI 1:	User interface	The system shall have an intuitive and simple user interface *Comment: The most important functions in the system should be possible to learn in one day, and the efficiency on the working habits of the actor shall not decrease*
NFI 2:	User interface	The system shall use consistence in display and symbols *Comment: All menus shall be built on the same structure. Colours/ fonts should be similar on all pages*
NFI 3:	User interface	The system shall be accessible in both Norwegian and English *Comment: Not all employees at the hospital speak fluent Norwegian*
NFI 4:	User interface	The system shall have user interfaces adjusted to the different roles *Comment: Not all actors in the system shall be able to perform the same functions. Different user interfaces are therefore needed*
NFI 5:	Communication interface	The SAPT shall be able to communicate with the Map System by means of HTTP
NFI 6:	Communication interface	The SAPT shall be able to communicate with the Object Database by means of JDBC
NFI 7:	Communication interface	The SAPT shall be able to communicate with the Location Server by means of JMS
NFI 8:	Communication interface	The SAPT shall be able to communicate with the Catalogue Service. The interface is not yet specified
NFI 9:	Communication interface	The SAPT shall be able to communicate with the Message Service. The interface is not yet specified

Table 7–20: Non-functional requirements to interfaces

7.4.2 Performance requirements

Performance requirements specify the performance of a system. The performance requirements applying to the SAPT is stated in table 7–21.

Requirement:	Type:	Description:
NFP 1:	Capacity	The system shall be able to handle >50 000 units *Comment: All equipment, staff, etc., related to the hospital shall be included*
NFP 2:	Response time	The system shall give response maximum 5 seconds after a request is sent
NFP 3:	Accuracy	The system shall be able to give positions with an accuracy at least on room level
NFP 4:	Life time - tags	The batteries in tags shall last at least 5 years

Table 7–21: Non-functional requirements to performance

7.4.3 Technical requirements

Technical requirements specify the requirements to the hardware and software of a system. The technical requirements applying to the SAPT is stated in table 7–22.

Requirement:	Type:	Description:
NFT 1:	Hardware	The system has to be run hardware that satisfies the requirement to response time (NFP 2)
NFT 2:	Hardware	The system shall be accessible from any normal PC/PDA/other terminal
NFT 3:	Software	The system shall be accessible from any terminal with an Internet connection. Comment: *This relates to actors logged in with roles 1-3*
NFT 4:	Software	For actors logged in with role 4, the system shall be accessible from specific terminals with specific software *Comment: The specific software will consist of a Java application*

Table 7–22: Non-functional requirements to technical solutions

7.4.4 Quality requirements

Quality requirements specify the quality-level of a system. To cover the quality requirements to the system, a closer look will be taken on user requirements, cost, availability, dependability, security and maintenancebility. The quality requirements applying to the SAPT is stated in table 7–23.

Requirement:	Type:	Description:
NFQ 1:	User requirement	The users of the system shall be familiar with and know how to use a normal browser *Comment: The actors has to be able to use a browser, and have an understanding for the principle with buttons, links, text-areas, etc.*
NFQ 2:	User requirement	The actors in the system shall be registered in the Catalogue Service *Comment: Only authorized actors shall have access to the system*
NFQ 3:	Cost	The tags used in the system shall be small and cheap *Comment: This relates to the large amount of tags that are needed*
NFQ 4:	Availability	For actors logged in to the system with roles 1-3, the system shall be available from the Internet
NFQ 5:	Dependability	The system shall have an up time of 99.9%
NFQ 6:	Dependability	The system shall not have more than one instance of the same object (be ambiguous)

Table 7–23: Non-functional requirements to quality

Requirement:	Type:	Description:
NFQ 7:	Dependability	The system shall be able to handle a power failure without loosing data
NFQ 8:	Maintenancebility	The system shall have a documented source code to ease maintenance when this is needed
NFQ 9:	Maintenancebility	It shall be possible to upgrade or expand the system in a modular way
NFQ 10:	Security	The system shall authenticate all actors with username and password
NFQ 11:	Security	Actors not authorized should not obtain access to the system
NFQ 12:	Security	The system shall limit the access to sensitive information, so that the different user groups only gets access to relevant information *Comment: Not everybody should get access to all patient's journals, etc*

Table 7–23: Non-functional requirements to quality

7.5 Prototypes for the user interface

To concretise and visualize the requirements, it was decided to make some prototypes of the PC interfaces for the system. These prototypes are meant as examples, and as a goal to work towards when the system is to be implemented.
These prototypes are made in a tool called Microsoft PowerPoint [36]. This means that the examples shown here are just modelled (drawn), and is *not* implemented yet.

7.5.1 General structure

It has been decided to let the interface be structured as shown in figure 7–5. The different parts of the figure are explained below.

- *Header:* Will contain things like the logo of the system, the name of the system, etc.
- *Footer:* Will contain contact information to the web master of the pages, information on when the pages where last updated, etc.
- *Site:* This is the part of the interface that gives textual information to the actor. All searches, parameter inputs, etc will take place here. The content will vary as the actor manoeuvre between the different menus. Error messages and verification messages are also shown here.
- *Control:* Will contain buttons to manoeuvre between the different menus, such as a 'main page' button, a 'back' button, etc.
- *Map:* This part will contain maps.
- *Buttons:* Will contain buttons relating to map functions, for instance buttons to zoom in and out of the map, buttons to move the map between different floors, etc.

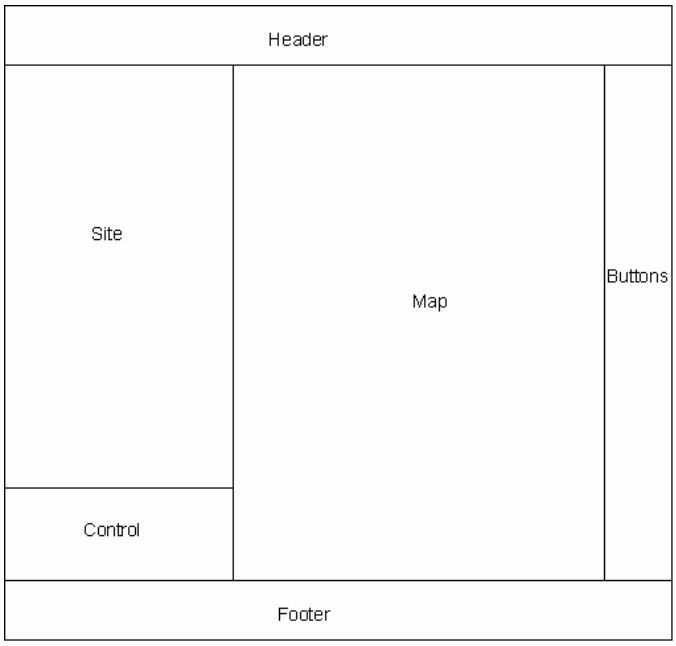


Figure 7–5: Example structure for the user interface

7.5.2 Examples of web interfaces

This section will show some examples of user interfaces intended for the web. Actors in the system with role 2 and 3, will access the system by means of a web interface.

The first example depicts how a web page that makes it possible for actors to search for objects might look. This is shown in figure 7–6. The map shown could be an overview map of the hospital.

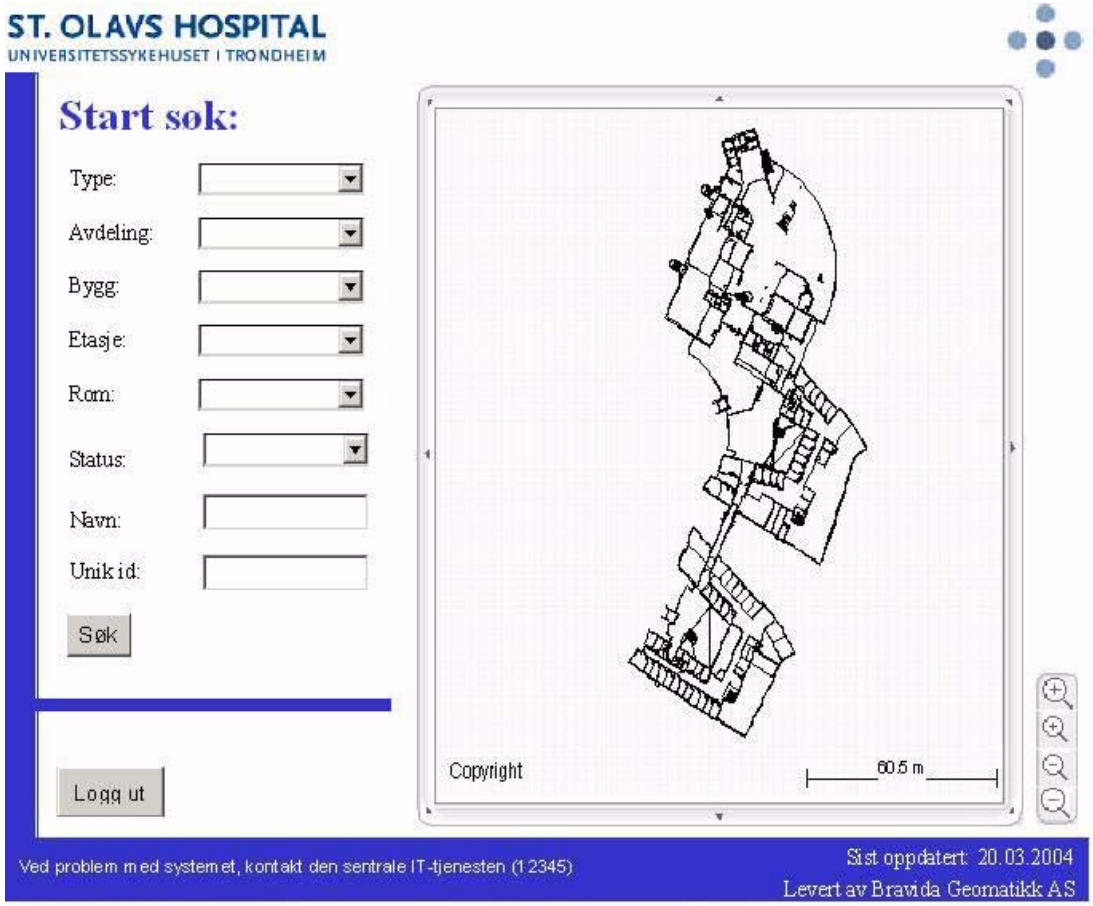


Figure 7–6: Web interface example for a search page

The next figure, figure 7–7, depicts how a web page that shows the positions of objects after a search can look like.

The figure shows that there were two objects that matched the performed search. An indoor map representing a part of the hospital is shown, and the two objects are drawn on the map. Here, they are shown with red circles, but it can be used different symbols depending on the object shown. As an example, a bed symbol could represent a ‘bed-object’. The button-row at the upper right hand in the figure shows that the objects are located on the first floor (of a total of four floors) in the building. Since no other of the buttons are clickable, there are no objects matching the search on other floors.
The figure also shows a way to realize the functional requirement that an actor shall be able to get information on objects. The actor can here uniquely select one of the objects, and then press a button to get a list of information relating to it. It should also be possible to click on the object-symbols in the map, and get some information on the object. This could for instance be the objects unique ID, its current status, the room number of its current location, etc.

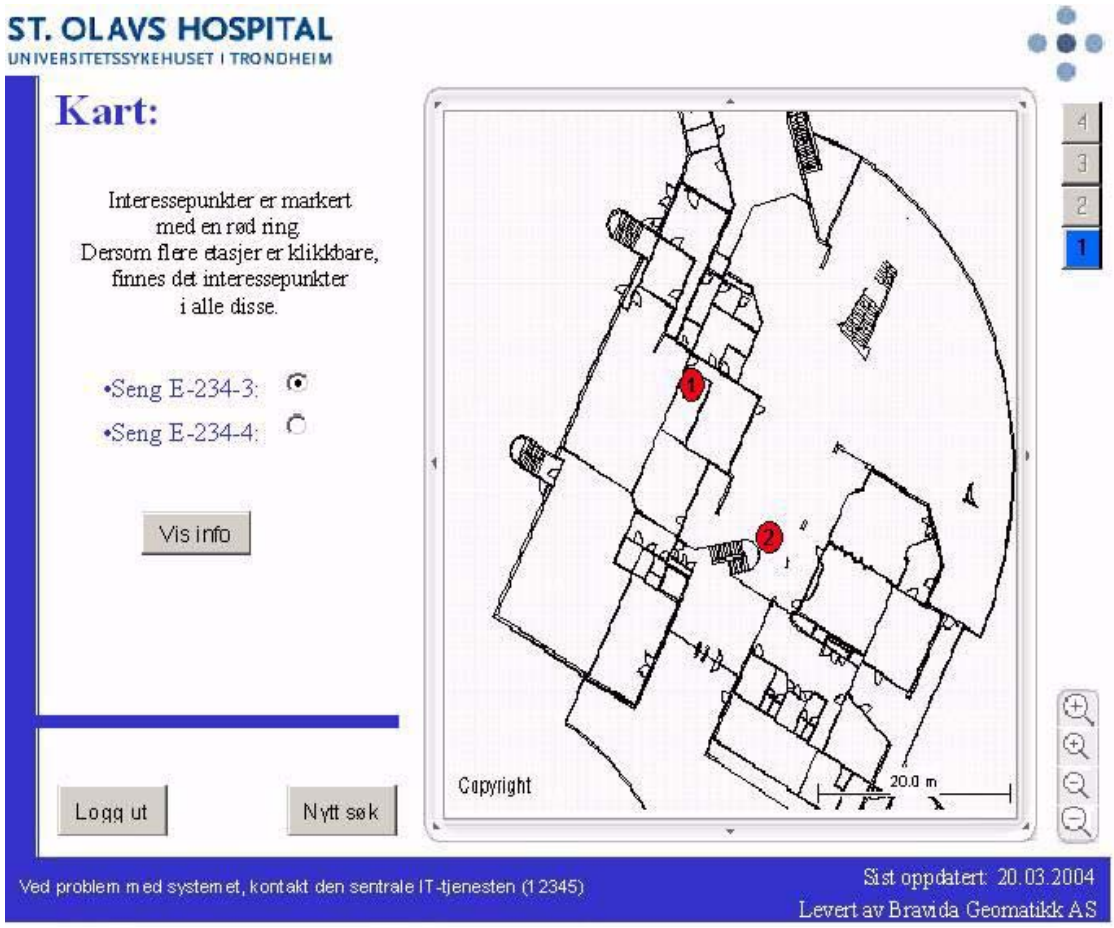


Figure 7–7: Web interface example for a page showing objects on a map

7.5.3 Example of Java application interface

This section will just introduce how an interface intended for a Java application might look like. Actors in the system with role 4 shall access the system by means of a Java application interface.

An example is shown in figure 7–8.

It has been emphasized that the Java interface should resemble the web interface, so that actors can use both interfaces without having to learn a completely new system.

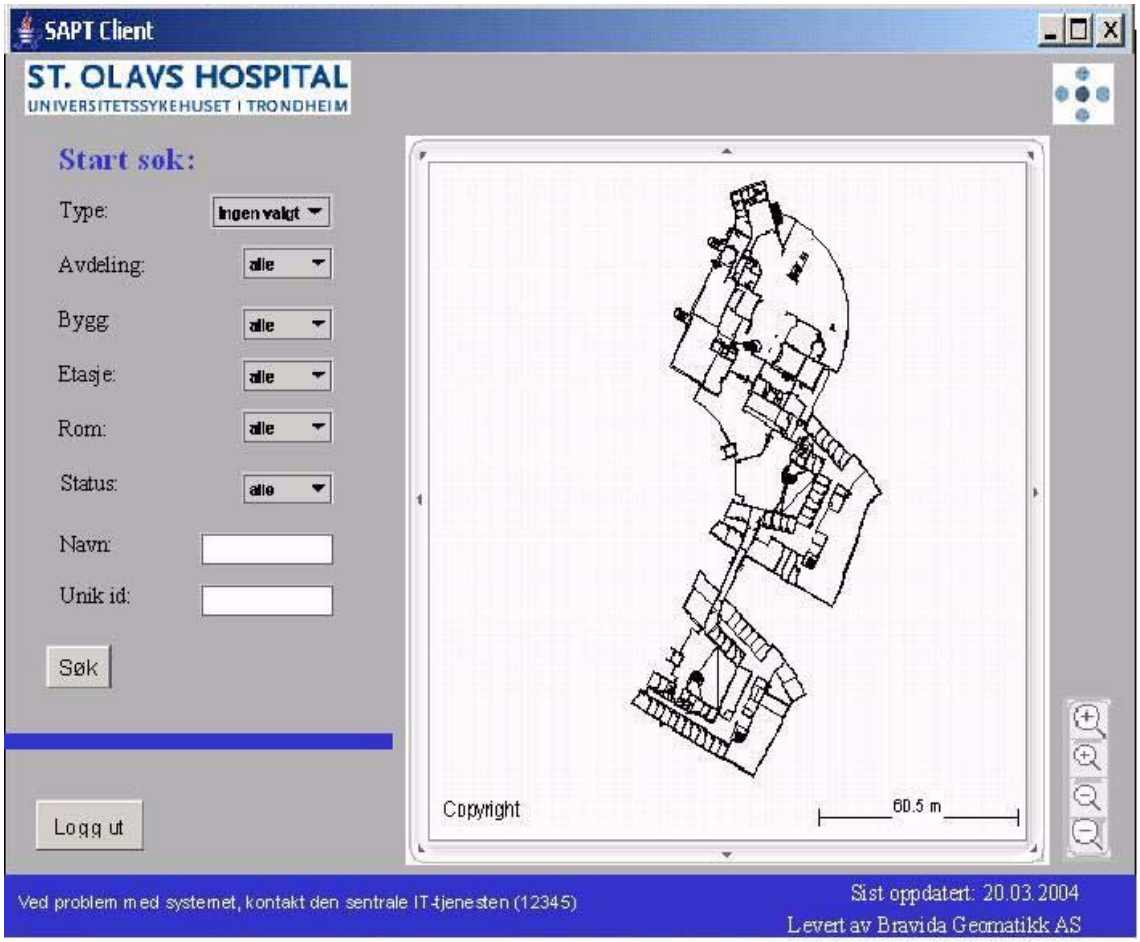


Figure 7–8: Java application interface example for a search page

8 Functional design

The purpose of this chapter is to describe the overall behaviour of the system. It will specify precisely the system's functional properties. The solution presented here is designed in close relation to the requirements specification. This to be sure to fulfil all the functional requirements depicted for the system.

It is in the nature of a functional design to model the system on an abstract level from where many alternative implementations are possible. In fact, one purpose of the functional design is to provide a basis for selecting the best possible implementation among many alternatives. Consequently, there will be several possible implementations corresponding to each functional design.

8.1 Overall architecture

Figure 8–1 gives the overall architecture for the system. It shows the components of the system, and how they are connected to each other. It is developed from the basic figure 3–1 on page 9.

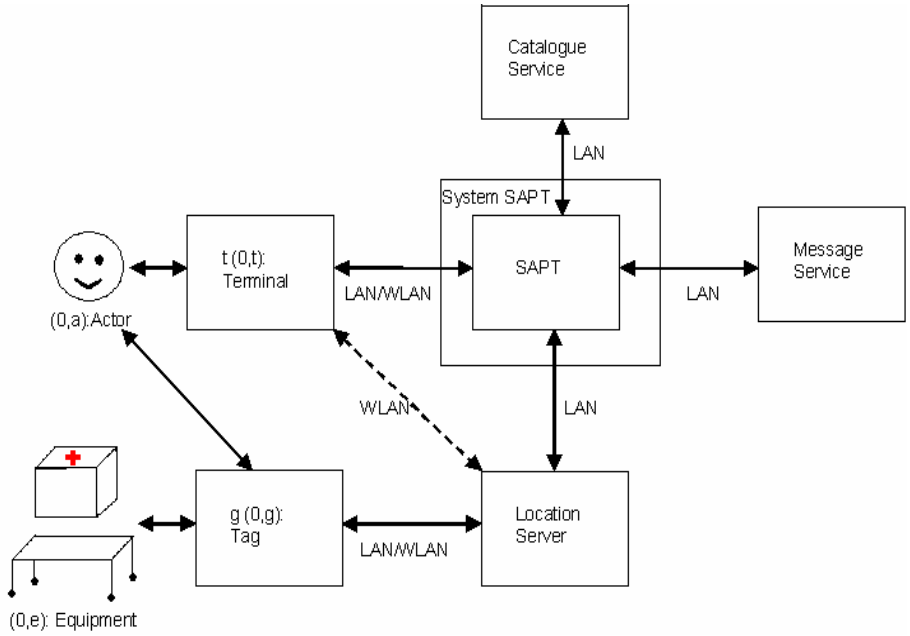


Figure 8–1: Overall architecture of the SAPT

As depicted in the figure, actors can access the system from terminals. There will be between 0 and ‘a’ actors in the system, and between 0 and ‘t’ terminals. This is not a one-to-one relationship, because actors can share terminals, have more than one terminal, etc. There will also be between 0 and ‘e’ different equipment in the system, and some of these will have a tag attached to them. Some of the actors will also possess a tag. The will exist between 0 and ‘g’ tags in the system. The tags will communicate with the Location Server via LAN or WLAN. If WLAN positioning is used, some of the actors will rather use their WLAN equipment (laptop/ PDA/etc.) to be positioned than a tag. Hence the dotted line.
There will only exist one logical instance of the following entities: the System SAPT, the Location Server, the Catalogue Service and the Message Service. This means that these entities can be replicated in hardware, but that the actors in the system will experience them as just one logical entity.
It has been chosen to model the Map System and the Object Database as a part of the System SAPT, as they only communicate with the SAPT and not with any other components.

8.2 System blocks

This section will describe the component ‘System SAPT’ in figure 8–1 in more detail. It will use the notation of SDL (Specification and Description Language, see [32] or [37]) in the description.

A top-down design method is used, which involves looking at the overall problem (specifying the total system) on a high level of abstraction, and then decompose this big problem in a number of smaller sub-problems. Each one of these sub-problems is then later studied on a lower level of abstraction. If necessary, these sub-problems can also be decomposed into even

more detailed sub-problems. This process is repeated until the sub-problems are so small that they are trivial to solve.
The top-down design method makes it possible to solve the overall problem step by step, instead of directly making a complete solution. [38]

An overview of the 'System SAPT' is given in figure 8–2.

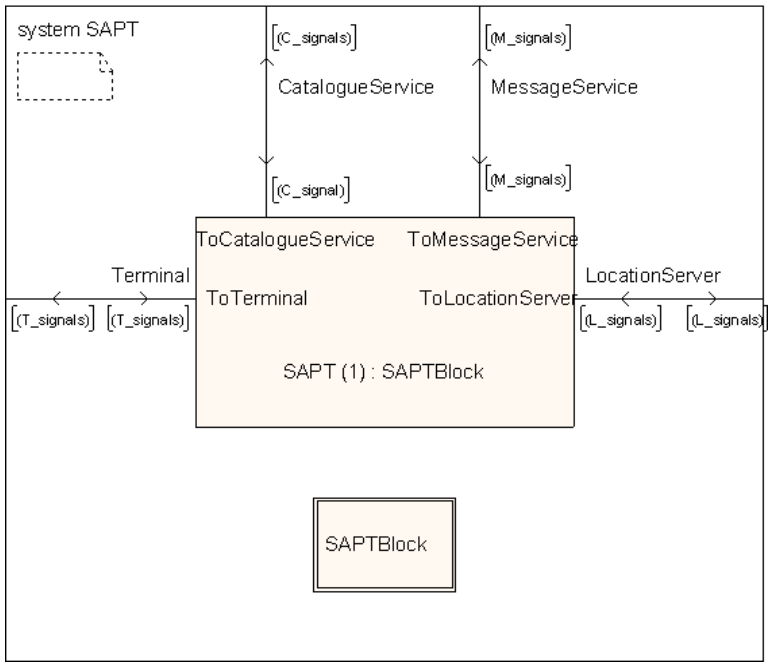


Figure 8–2: SDL overview of the SAPT

The figure shows the System SAPT, and its connecting communication channels. There will be one logical channel for each of the different components it can communicate with (but these might actually be located in the same physical cable). The channels will typically consist of ethernet cables.
The channels labelled 'CatalogueService', MessageService' and 'LocationServer' will connect to these components respectively. The components were described in chapter 5: 'Surrounding systems'. The channel labelled 'Terminal' will communicate with the actors terminals.

The inside of the 'SAPTBlock' in figure 8–2 will look like depicted in figure 8–3.

The SAPTBlock consists of the block 'SAPT', along with the blocks 'OIDatabase' and 'MapSystem'. The blocks 'OIDatabase' and 'MapSystem' is, as mentioned before, already implemented, and will therefore not be decomposed further here.

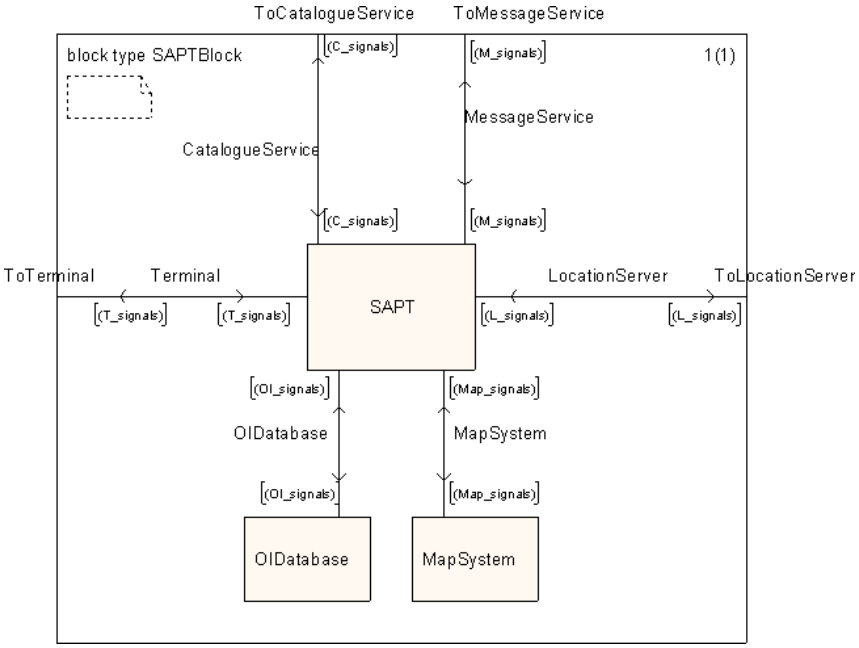


Figure 8–3: Inside the SAPTBlock

When decomposing the block SAPT, it has been found that several more blocks are needed to handle the functionality depicted in chapter 7: 'Requirements specification'. This is shown in figure 8–4.

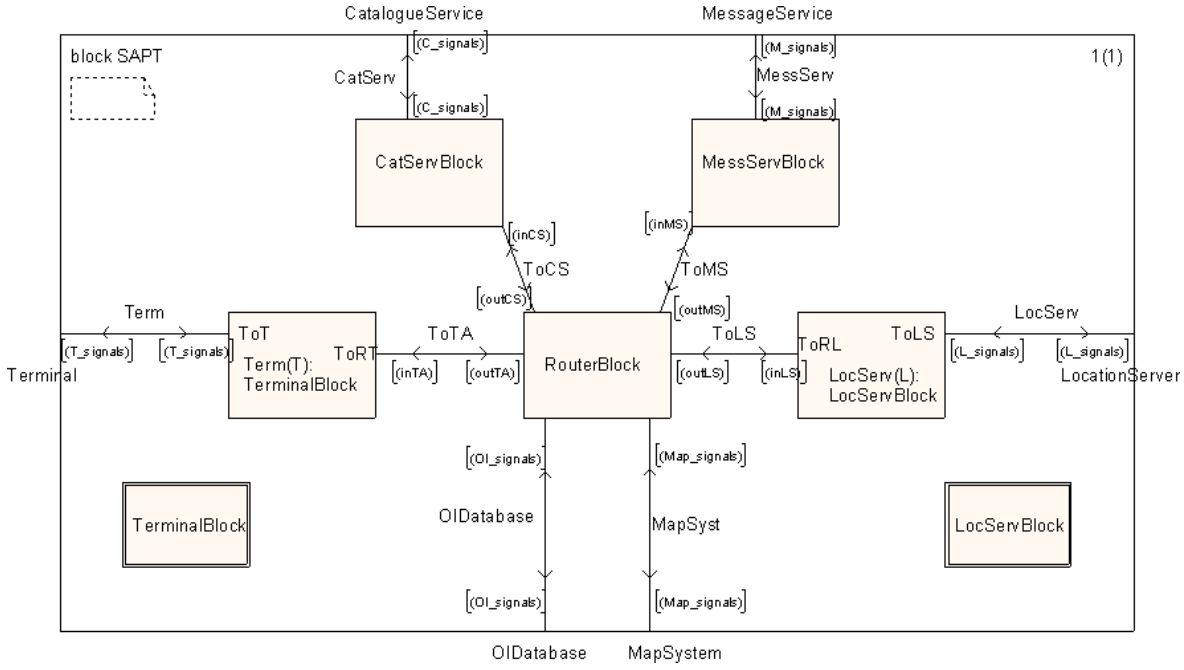


Figure 8–4: The SDL blocks inside the SAPT

The reason for having different SDL blocks to handle the various interfaces for the Terminal, the Catalogue Service, the Message Service and the Location Server is that each of these interfaces can give input to the system independently of the others. By allocating a separate block to each of them, the system will be able to handle these inputs in parallel.

The reason for not having separate blocks for the Object Database and the Map System is that no independent messages will ever originate from these. The only messages they will send out are responses to request originating at other processes. The only thing a 'Map-block' for instance would do, is to act as a router for messages to and from the Map System. No actual logic would be performed, and the block is found to be redundant.

The decisions made here are in accordance with the rule for concurrency from [32], which states that one shall '*Model independent and parallel behaviours as separate processes*'.

The processes inside the blocks on figure 8–4 are described in the next section.

8.3 System processes

A decision has been made to let the block that first receives a request to manage all further processing that is necessary. For instance, if the TerminalBlock receives a request to search for an object, it is this block that asks the Object Database for location of the object, the Map System for maps, and makes sure the actor gets back the proper response.

An informal class diagram has been composed to indicate which functions the different processes will need to handle from the environment. Class modelling is a specialized type of modelling concerned with the general structure of a system. It is usually used during analysis and design activities [34]. The diagram is shown in figure 8–5.

Note that the figure does not show all functions that the different blocks need to be able to handle. It only shows the functions that they will be the first ones to receive.

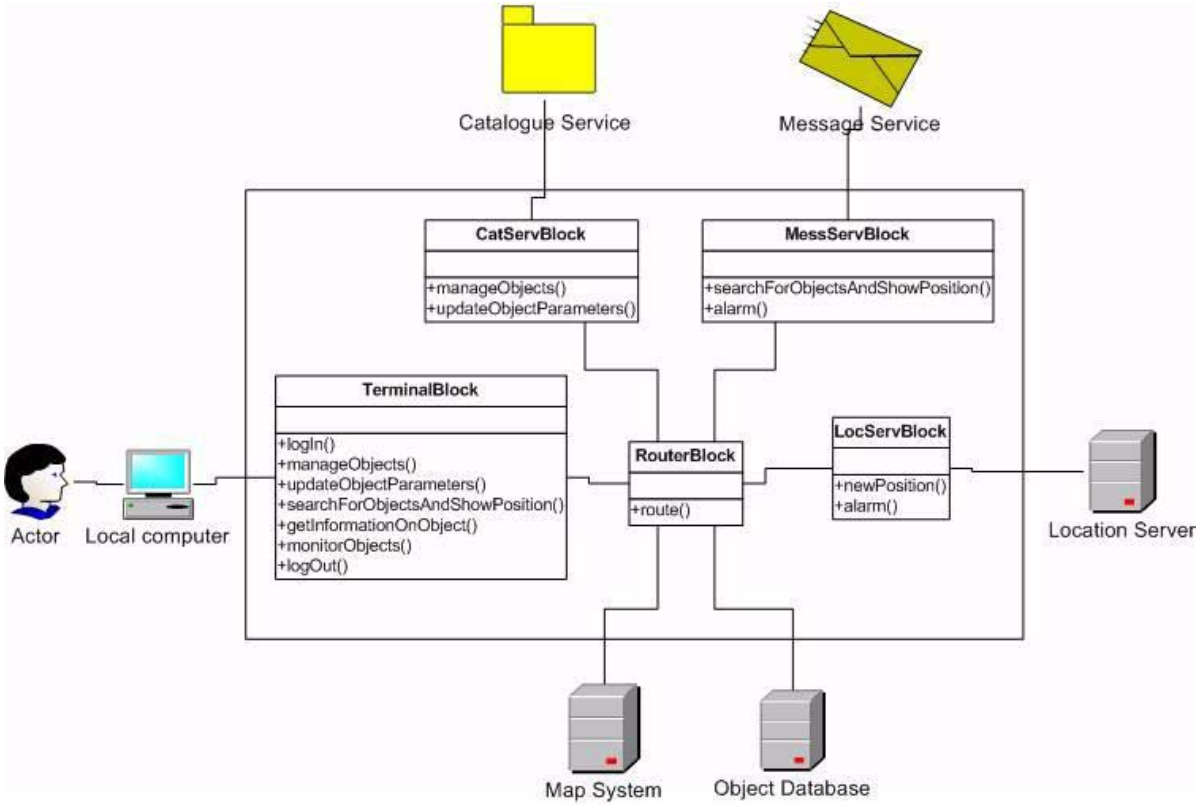


Figure 8–5: Model indicating method handling by blocks

8.3.1 TerminalBlock

The TerminalBlock will contain one process, called TerminalAgent. This is shown in figure 8–6.

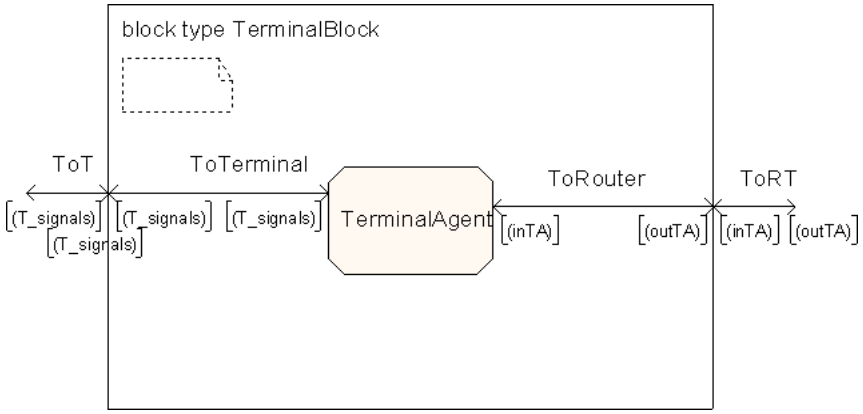


Figure 8–6: Process inside the TerminalBlock

The TerminalBlock is actually a block set, which is instantiated to a certain number. Each block instance will be responsible for the control of services being accessed by a single actor. In figure 8–4 on page 60, the TerminalBlock is shown to have T instances, i.e. a fixed number. When this is a fixed number, an actor that wants to start using the system is assigned one of the instances from a pool of instances. If there are no more free instances, the actor will perceive the system as being occupied/out of order. A better solution for this is that each time an actor wants to start using the system, i.e. start up a terminal application, a new instance is created. Then there will not be a problem with the system running out of instances.

The reason for having several instances of the TerminalBlock is that it needs to handle all the different user sessions that can exist simultaneously. One TerminalBlock is assigned to each user session. The TerminalAgent process inside the TerminalBlock can store data between transactions for the current session. When a session is terminated, i.e. when the actor logs out of the system, this instance is also terminated.

8.3.2 RouterBlock

The RouterBlock will contain one process, called Router. This is shown in figure 8–7.

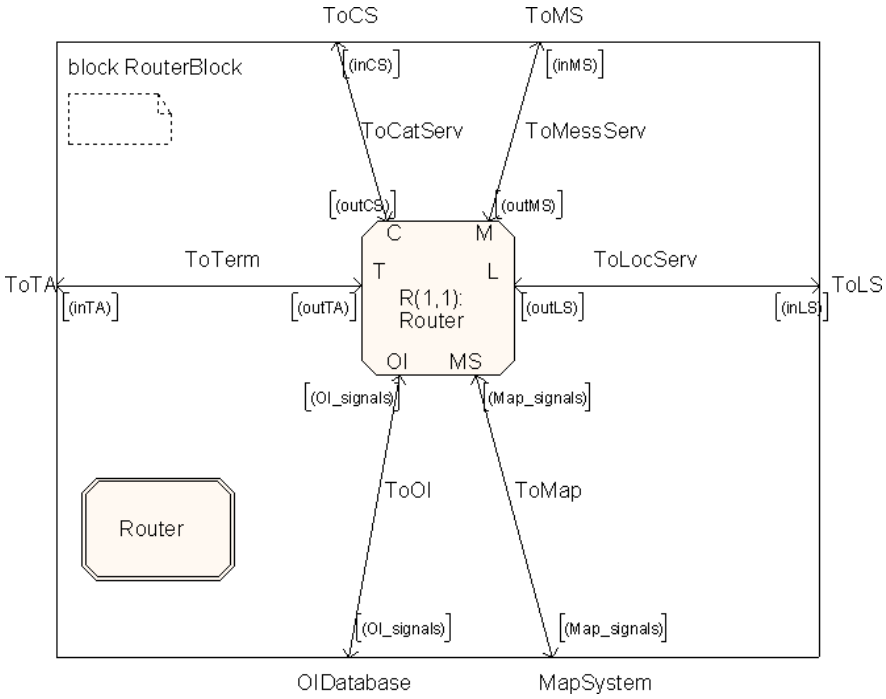


Figure 8–7: Process inside the RouterBlock

The RouterBlock will handle all communication between the other blocks. The SAPT will most probably be a distributed system, and the RouterBlock will help realize localisation independence. A process should be able to send a signal to another process and to receive a response without knowing the physical localization of the process.

It will exist only one logical instance of this block in the system at any time. The component is constructed very simple, as it will never contain any state or session information. It is completely stateless.

By the introduction of the Router process, a two-level identification scheme can be realized. The idea is that a process that has a message to send, only needs to know the location of the Router. The process sends the message to the Router, with the final destination transferred as a parameter in the signal. The Router then forwards this message to the proper destination. This is in accordance with the rule for routing processes from [32], which states that *'For each block where there is a choice between local and remote communication, use a two-level addressing scheme and hide the routing knowledge in a routing process'*.

8.3.3 LocServBlock

How to design this component was a difficult task. Whether to have one block instance for each object at the hospital or not was a question that had to be discussed. It was found that the functional design would be easiest if logically, one block instance was assigned to each of the potentially over 50 000 objects to be positioned at the hospital.
The inside of the LocServBlock is shown in figure 8–8.

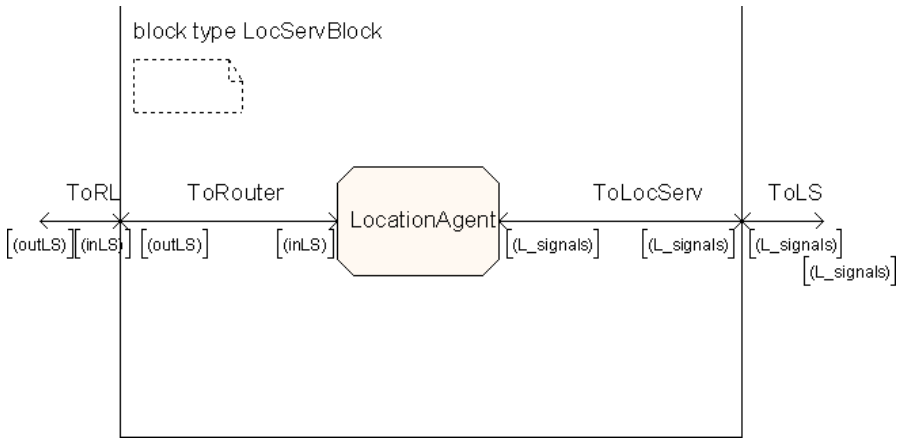


Figure 8–8: Process inside the LocServBlock

To have one separate block instance handling each object eases the management of objects. It was decided that when a new object is added to the system by invoking the method 'manage objects', a new instance of the LocServBlock is created. Each object then has its own LocationAgent process, which can be in a particular state independent of all the other processes. This makes detecting alarm for an object is easier, for instance, as each LocationAgent is concerned with only one object.
When an object is deleted, the LocationAgent process for this object is also deleted.

This decision is also in accordance with the rule for concurrency mentioned at the end of section 8.2.

8.3.4 CatServBlock

It will only exist one logical instance of the CatServBlock in the system. It has been designed to contain two different types of processes, as shown in figure 8–9.

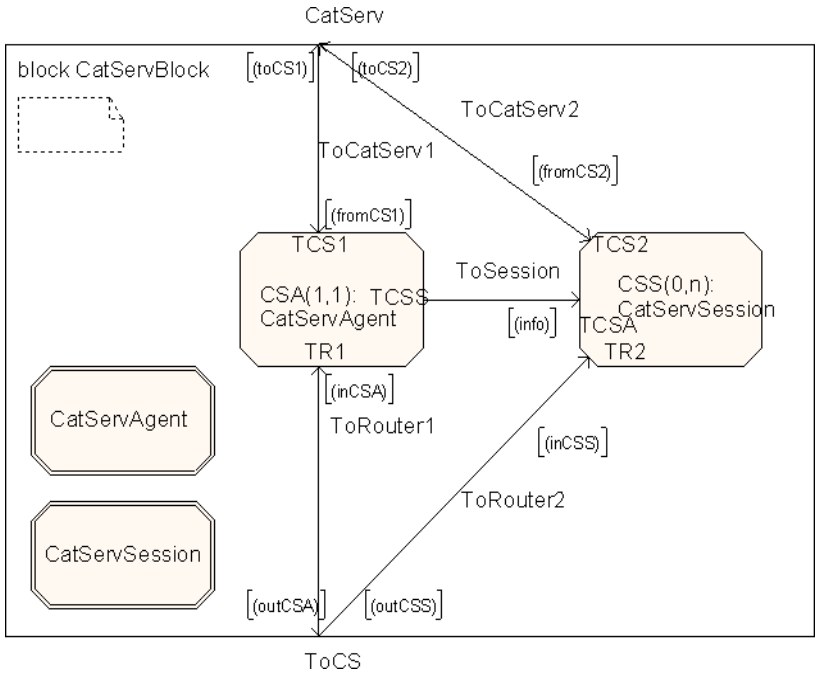


Figure 8–9: Processes inside the CatServBlock

The CatServBlock will handle some requests originating from actors terminals, and some originating at the Catalogue Service. When a request arrives from an actor, the CatServBlock only needs to act as a router to forward the request, and can be completely stateless. The TerminalBlock for the actor will contain all state information for this session. CatServAgent is used for this purpose. The CatServAgent process will always run, and it exist only one instance of it.
When a request arrives from the Catalogue Service however, the CatServBlock will need to store state information for this request. This can for instance be to make sure that the Object Database is properly updated. It has therefore been decided that when the CatServAgent receives a request originating from the Catalogue Service, it will make a new instance of the process type CatServSession to handle this session. The lifetime of this process will be from it is created until a reply on the request is sent back to the Catalogue Service. The instance will then be deleted.

8.3.5 MessServBlock

The MessServBlock will handle some requests originating from actors terminals, and some originating at the Message Service. The behaviour of the MessServBlock will be logically equivalent to the behaviour of the CatServBlock, and it is therefore not described any further. The processes in the MessServBlock are shown in figure 8–10.

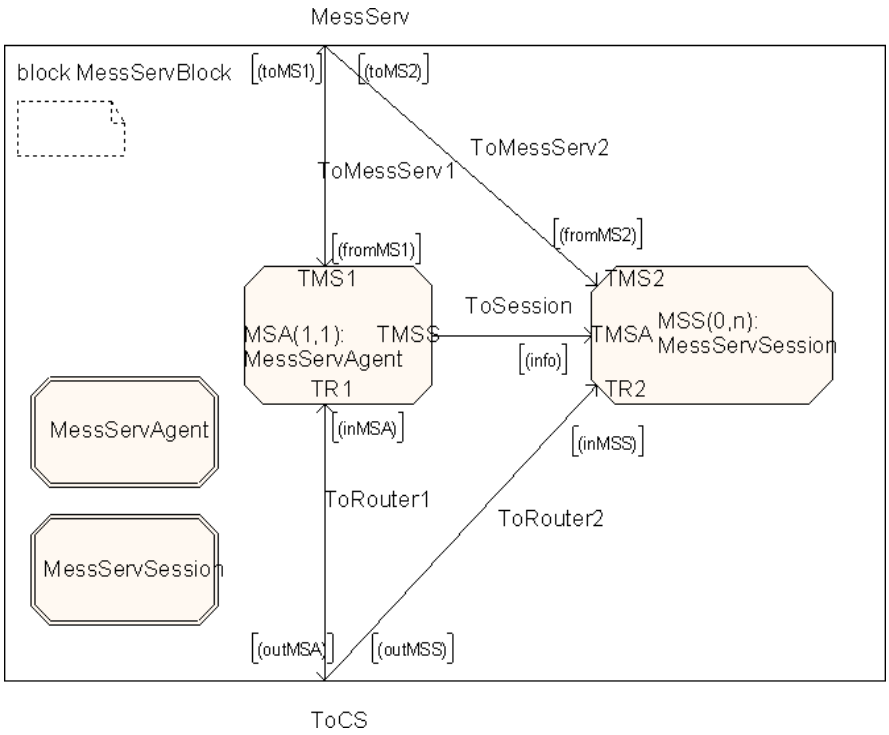


Figure 8–10: Processes inside the MessServBlock

8.4 Process interaction

The following part will describe in detail how the different processes shown in figure 8–6 to figure 8–10 will communicate with each other. The focus is on how the message flow will be in the system. It is based on the functional requirements relating to actors and to the SAPT.

The description will rely on the Message Sequence Chart (MSC) notation. MSC is a tool for specification and validation, which concentrates on describing the message sending between instances. [39]

A note regarding the MCSs shown here is that they only show the important parameters that are sent between the different components. They do in particular not show *all* the parameters that need to be sent, just a simplified view. In addition, the 'Router' process is not drawn in the diagrams. This is done to save space, but in reality, almost all messages sent between the different processes will pass through the Router.

8.4.1 Requirements FA 1 and FA 10

In this section, the following requirements will be described:

- FA 1: An actor shall be able to log in to the system
- FA 10: An actor shall be able to log out of the system

The sequence of actions will be as shown in figure 8–11.

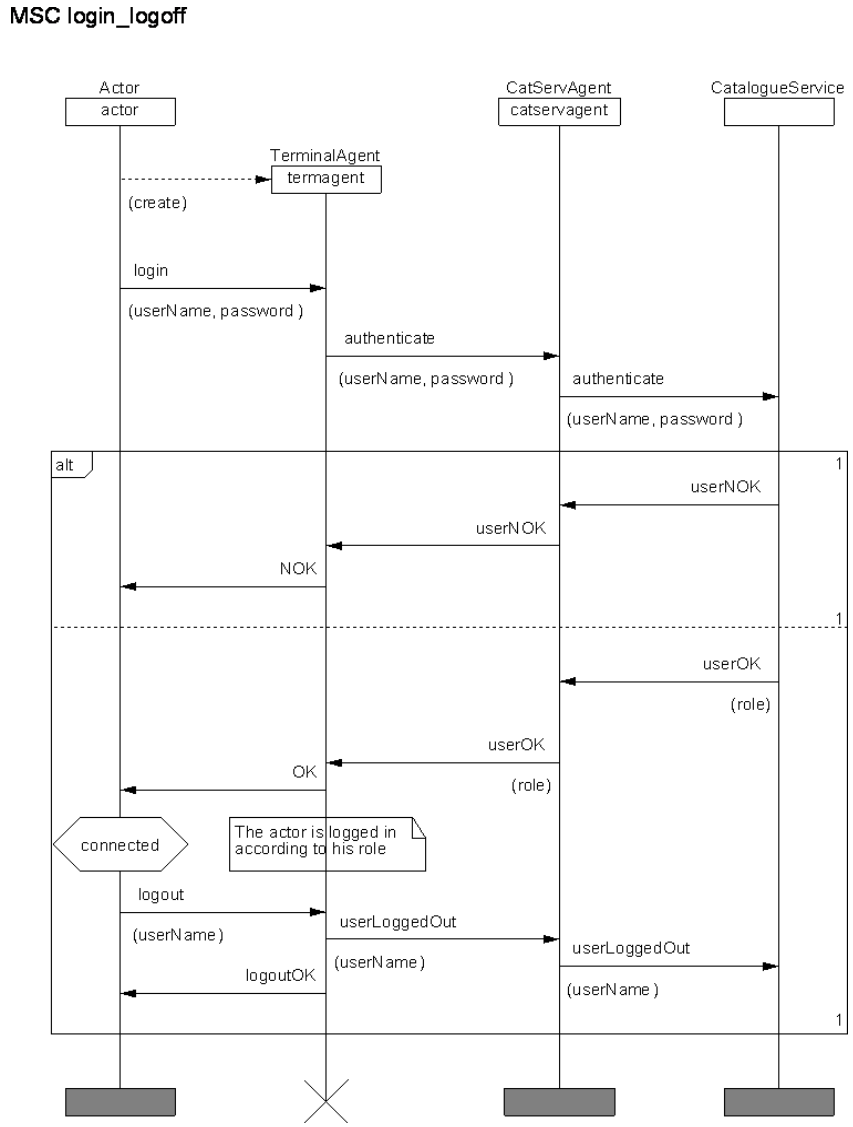


Figure 8–11: MSC for FA 1 and FA 10

For an actor to start using the service, he needs to log in to the system. To do this, the actor has to access the log in site of the system, and send a log in request. This will be done through a web page or a Java application at the actor's terminal. The actor encloses his username and password in the request, and sends it to the system, which then creates a new TerminalAgent instance for the actor. The TerminalAgent recognizes this to be a log in request, and forwards it to the CatServAgent, who then forwards the request to the Catalogue Service. The Catalogue Service authenticates the username and password, and if everything went fine, it sends back the role of this actor to the system. The actor is then logged in to the system according to his role, which involves that there will be functions available to execute all the use cases for this role.
A log out request disconnects the actor from the system, and deletes the current instance of the TerminalAgent.

A thing that has to be decided is if the Catalogue Service will keep track of who is logged in to the system at any time. If this is the case, the log out request from the actor needs to be sent all the way to the Catalogue Service, as shown in the figure. If not, the log out request can terminate at the TerminalAgent.

8.4.2 Requirements FA 2 and FS 1

In this section, the following requirements will be described:

- FA 2: An actor shall be able to manage objects
- FS 1: The system must be able to handle 'manage objects' calls from the Catalogue Service

These two scenarios are quite similar.
In figure 8–12 on page 69 the case that a request to manage objects originates at an actor is depicted. The call can be either to add a new object to the system, or to delete an object. If the actor wants to add a new object, he will have some information regarding the object that he wants to insert into the system. This information can for instance be submitted by the actor through filling out a form on his terminal. The new information is sent to the Object Database, which stores the object, and returns a unique identification number (objectID) for it. A new LocationAgent is created for the object, to handle all further calls to track it, set alarm on it, find it, etc. To keep the Catalogue Service up to date, a message is sent to it too, telling it to insert a new object with this unique objectID and information. As the actor at the terminal is not interested in knowing if the Catalogue Service is updated, the TerminalAgent handles this on its own, transparently to the actor.

To delete an object, the actor sends the objectID for the selected object to the system. The call is forwarded to the Object Database, which then removes this object from its records. The LocationAgent for the object is ended, and a message is sent to the Catalogue Service as well, telling it that the object has been deleted. This is done transparent to the actor.

Figure 8–13 on page 70 shows the case where a new object has been inserted or deleted in the Catalogue Service. To keep the SAPT and the Catalogue Service synchronized, this information needs to be replicated in the SAPT. When a request to do this arrives at the CatServAgent, it creates a new instance of the process CatServSession. All further processing needed to handle the request is done by this instance.

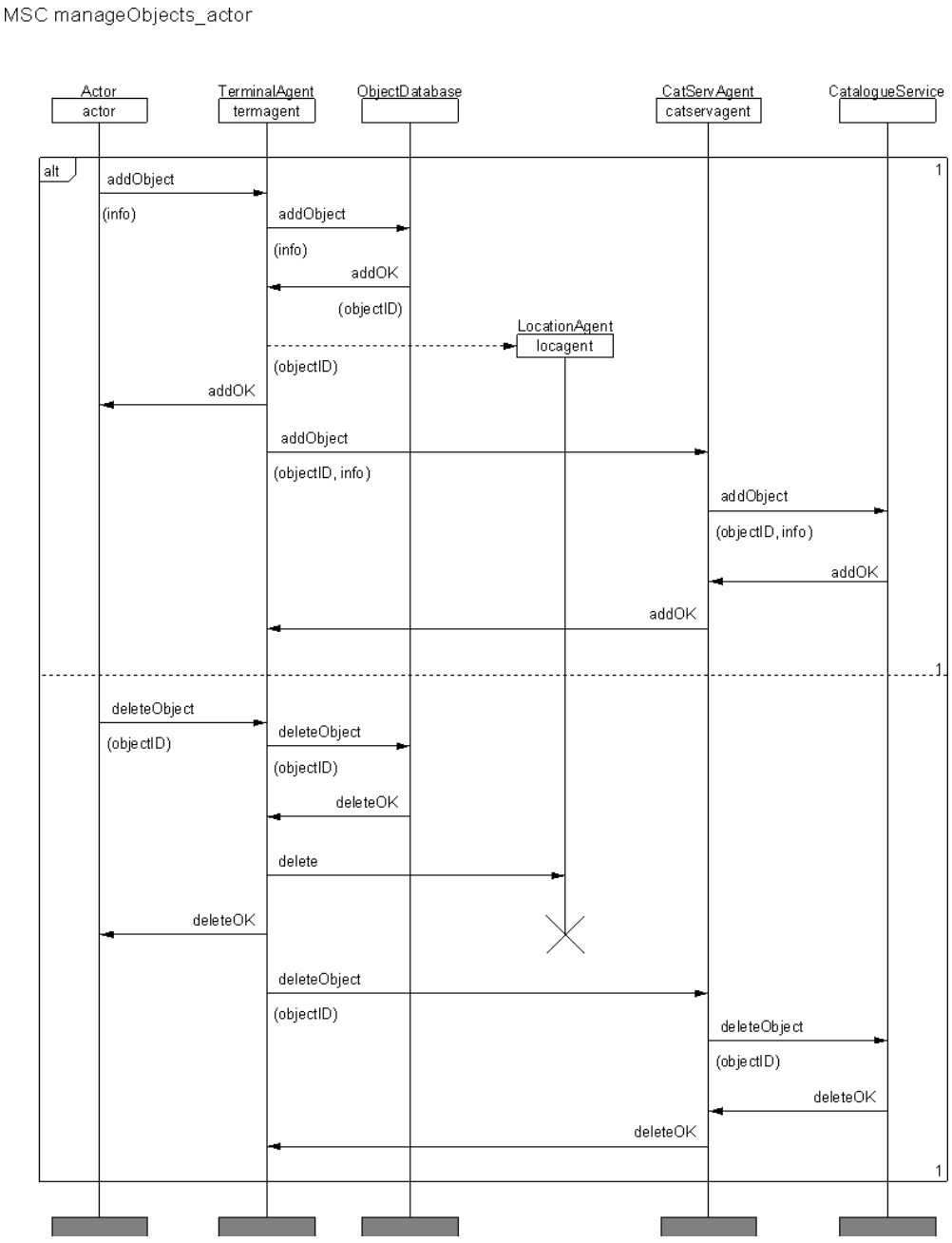


Figure 8–12: MSC for FA 2

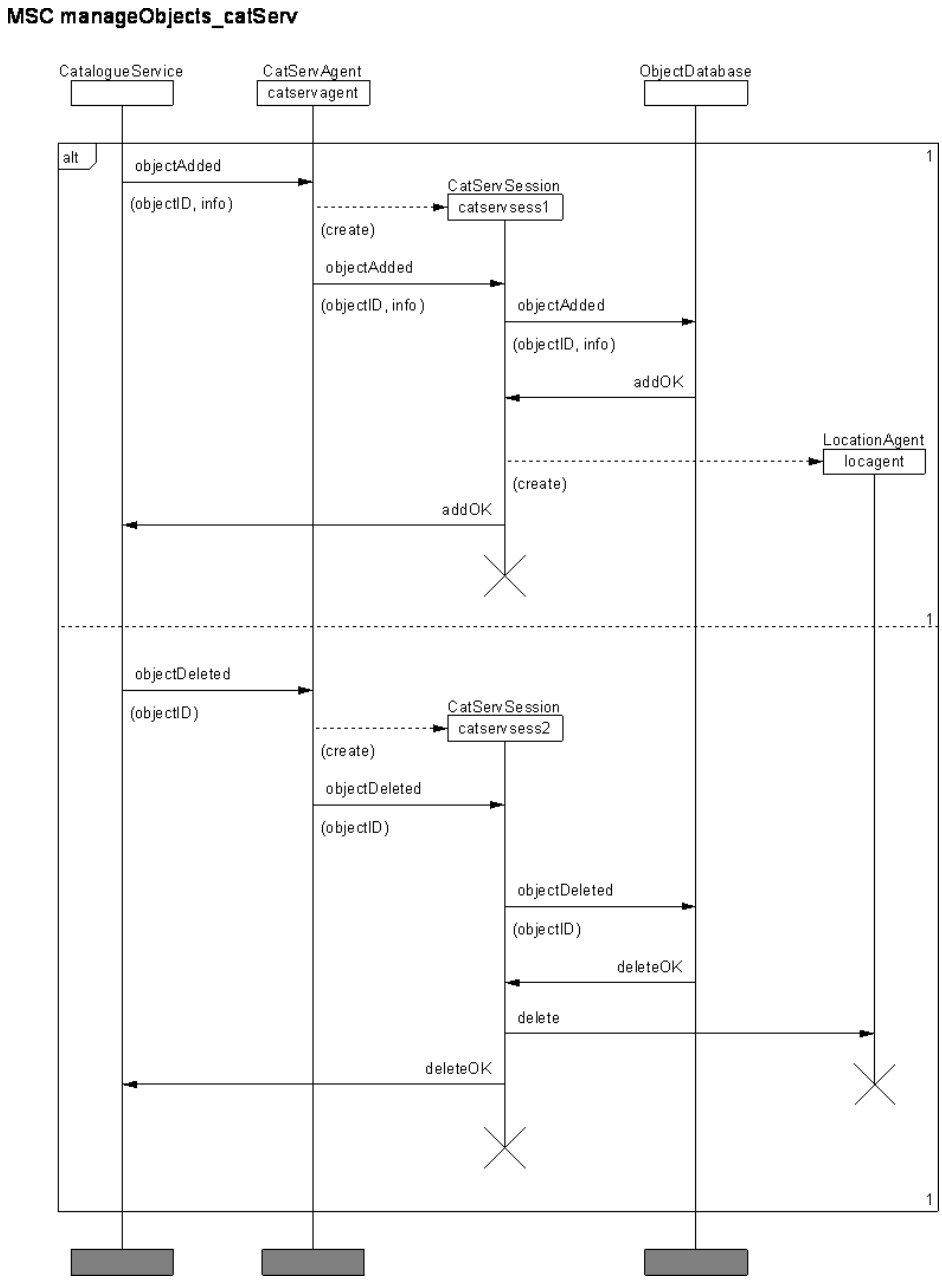


Figure 8–13: MSC for FS 1

8.4.3 Requirements FA 3 and FS 2

In this section, the following requirements will be described:

- FA 3: An actor shall be able to update object parameters
- FS 2: The system must be able to handle ‘update object parameters’ calls from the Catalogue Service

The sequence of actions will be as shown in figure 8–14 for FA 3, and in figure 8–15 for FS 2.

As mentioned before, the call to update an object's parameters can involve almost anything, for instance start/stop tracking, set/unset alarm, update attributes, change the zone the object belongs to, etc. Basically everything about an object that does not concern adding or deleting it.

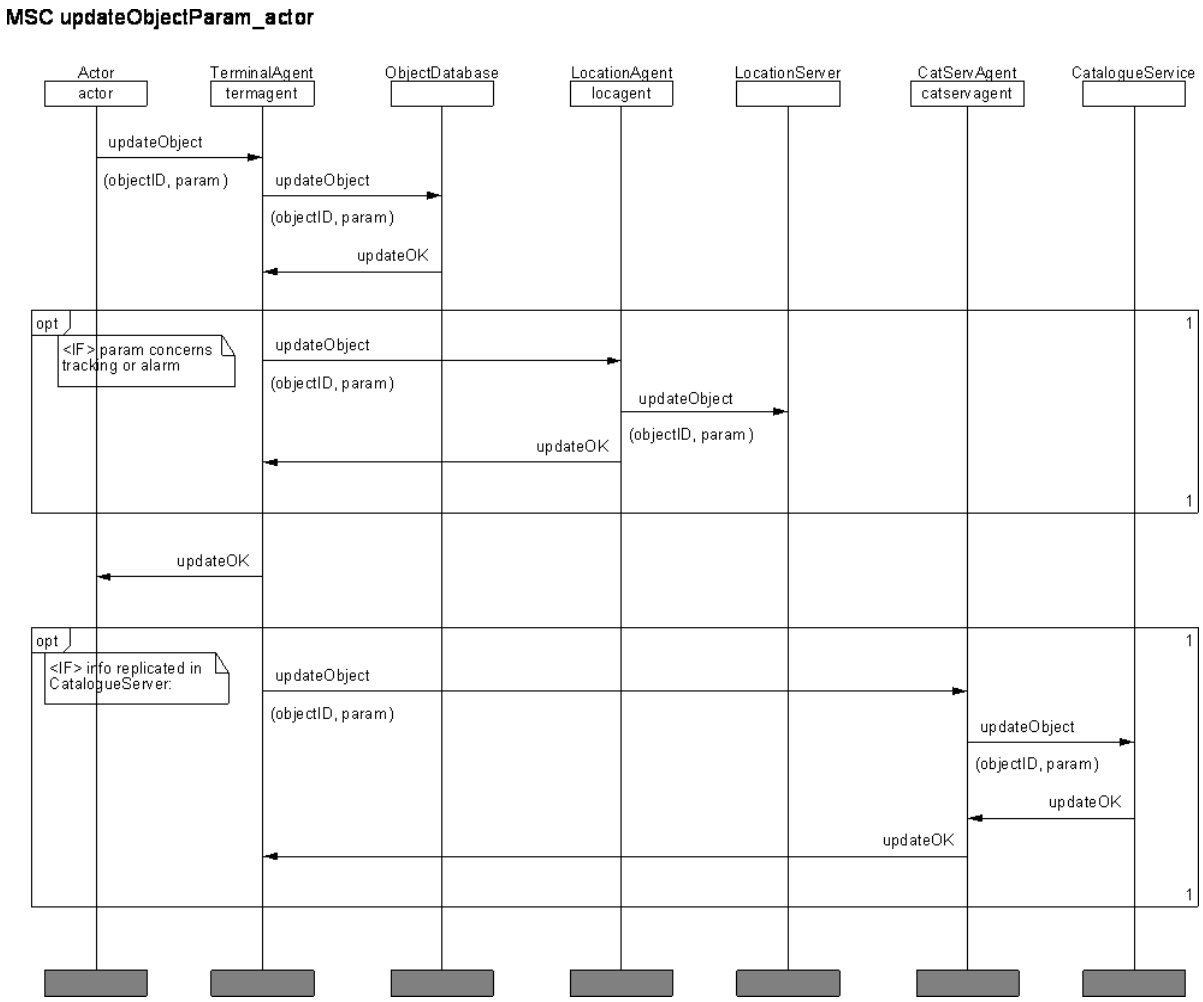


Figure 8–14: MSC for FA 3

These two scenarios are quite similar.
In figure 8–14 the case that a request to update an object's parameters originates at an actor is depicted. The information can for instance be submitted by the actor through filling out a form on his terminal, and is sent to the TerminalAgent of the actor. The information is forwarded to the Object Database, which is updated. If the new information is concerned with tracking, alarms, etc., the LocationAgent for the object is also updated and reaches the correct state. The proper action is forwarded to the Location Server. The problem of finding the correct LocationAgent for the selected object can be done by routing tables, for example.
The reason for updating the Object Database in any case is that it might be interesting to store information on whether an object is currently being tracked for instance.

If the information concerning the object is replicated in the Catalogue Service, a message containing the information is sent to it too. This is done transparently to the actor. Remember that not all information in the Object Database is replicated in the Catalogue Service and vice versa. The Catalogue Service does for instance not contain any information concerning the location of objects.

Figure 8–15 shows the case where some information relating to an object has been updated in the Catalogue Service, and this information is replicated in the Object Database. To keep the SAPT and the Catalogue Service synchronized, this information needs to be updated in the SAPT as well. When a request to do this arrives at the CatServAgent, it creates a new instance of the process CatServSession. All further processing needed to handle the request is done by this instance. Updating information originating at the Catalogue Service will never concern starting/stopping tracking, setting/unsetting alarm, etc.

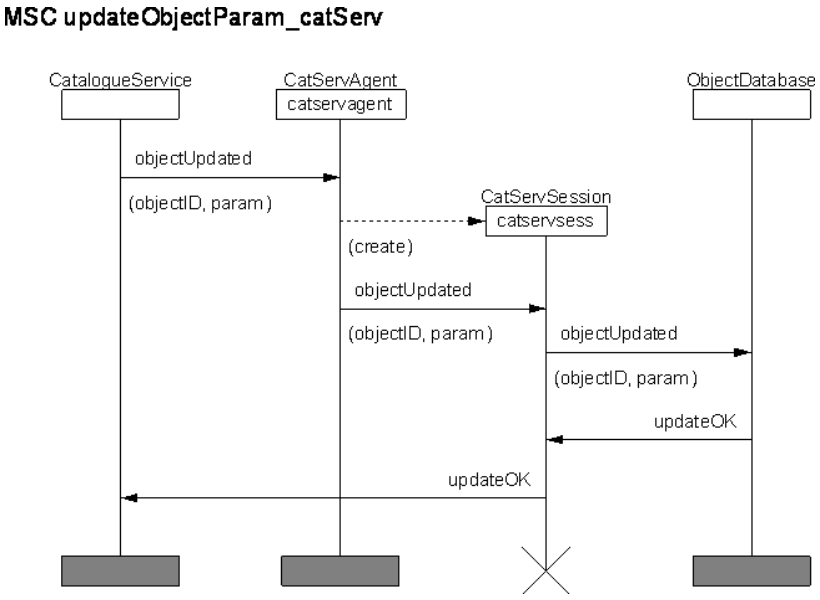


Figure 8–15: MSC for FS 2

8.4.4 Requirement FA 4

In this section, the following requirement will be described:

- FA 4: An actor shall be able to be positioned

The sequence of actions will be as shown in figure 8–16.

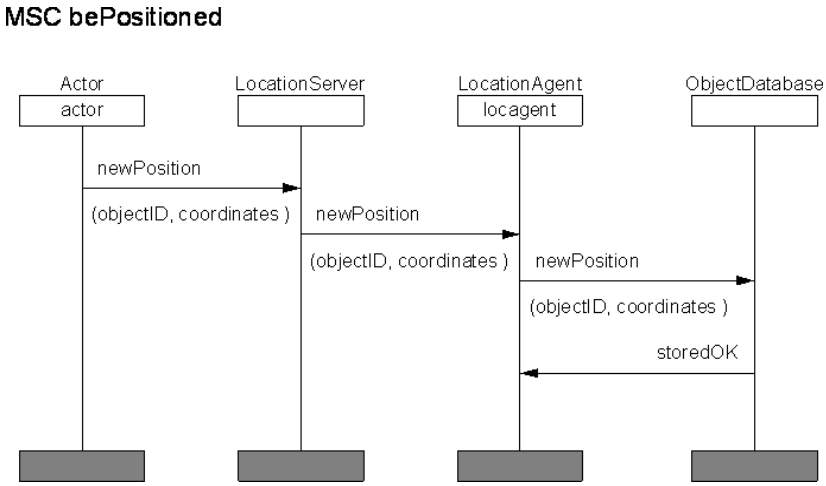


Figure 8–16: MSC for FA 4

Because this discussion has been concentrated around the functional requirements relating to actors, the figure shows what happens when an actor is positioned. The same thing happens however when any object is being positioned, so therefore the term ‘object’ is used in the following.

As mentioned in the textual use case ‘Be positioned’ in section 7.3.1.4, prior to that an object can be positioned, the object has to be added to the system. A LocationAgent process therefore already exists for this object. The object sends its new position to the Location Server, who then forwards it to the proper LocationAgent. The LocationAgent then makes sure that the Object Database is updated with the new position information.

8.4.5 Requirements FA 5 and FS 3

In this section, the following requirements will be described:

- FA 5: An actor shall be able to search for objects and show position of these
- FS 3: The system must be able to handle ‘search for objects and show position’ calls from the Message Service

The sequence of actions will be as shown in figure 8–17 for FA 5, and in figure 8–18 for FS 3.

These two scenarios are quite similar, except that if the request originates at an actor, there will potentially be many objects to show the position for, while if the request originates at the Message Service, only the location of one object is wanted.

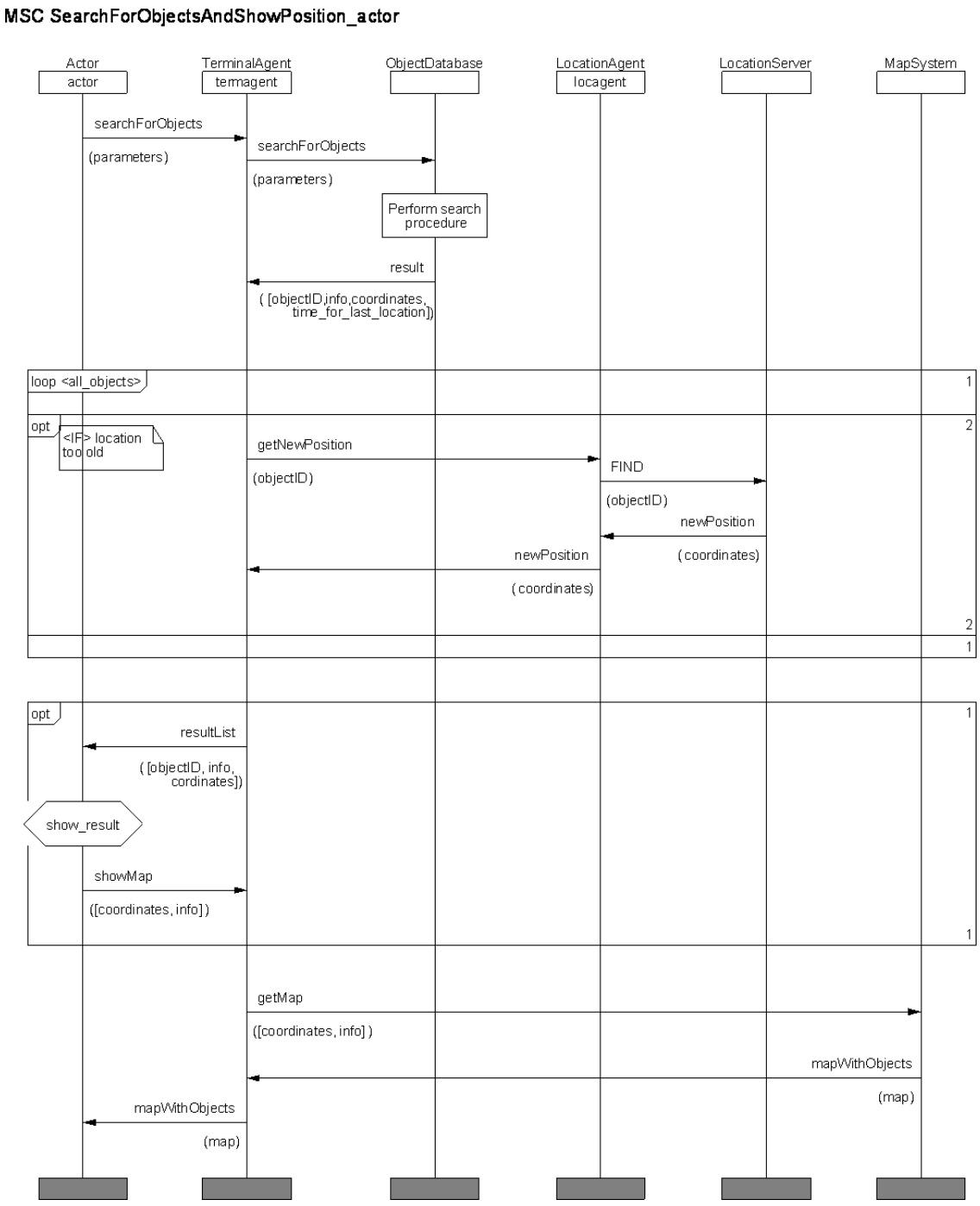


Figure 8–17: MSC for FA 5

Figure 8–17 shows the case when a request to search for objects and show the position of these originates at an actor. The request is forwarded by this actor's TerminalAgent to the Object Database, which performs a search based on the parameters. It returns a list containing all objects matching the search. Each object is returned with object ID, the last location registered for the object, the time that the last location was registered, and a little information on the object.

The TerminalAgent then checks all objects to see if the location registered for them is too old. For the objects with not new enough location information, the current locations for the objects are retrieved from the Location Server.
If the search did not return just one object (which for instance is the case if the actor did just input an object ID as parameter to the search), a list of objects matching the search is printed to the actor's terminal. If the actor inputs his current location as a parameter in the search, the objects closest to the actor can be shown first in the list. The actor can then select any number of the objects, and choose to view the location of these. A request is sent to the Map System, which returns a map showing the desired objects. Some information can also be shown in the map, such as the name of the objects and the room number where they currently are.
If there was only one object that matched the search, the map is shown straight away, and the actor does not have to choose from a list.

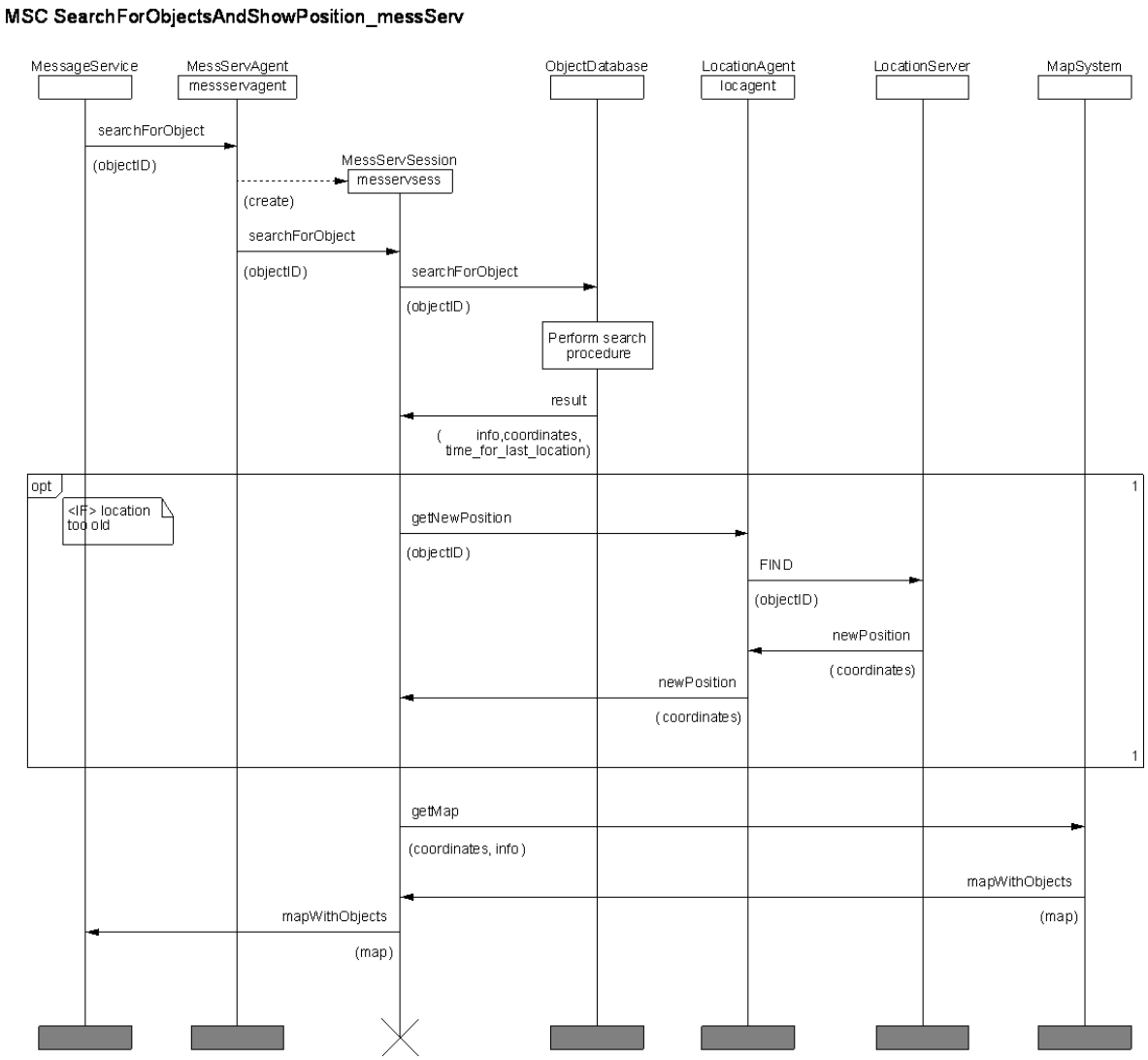


Figure 8–18: MSC for FS 3

When the request to search for objects and show the position of these originates at the Message Service, it will always be a request concerning only one object. This is shown in figure 8–18. When a request to do this arrives at the MessServAgent, it creates a new instance of the process MessServSession. All further processing needed to handle the request is done by this instance. The instance is terminated when the final reply for the request is sent.

8.4.6 Requirement FA 6

In this section, the following requirement will be described:

- FA 6: An actor shall be able to get information on objects

The sequence of actions will be as shown in figure 8–19.

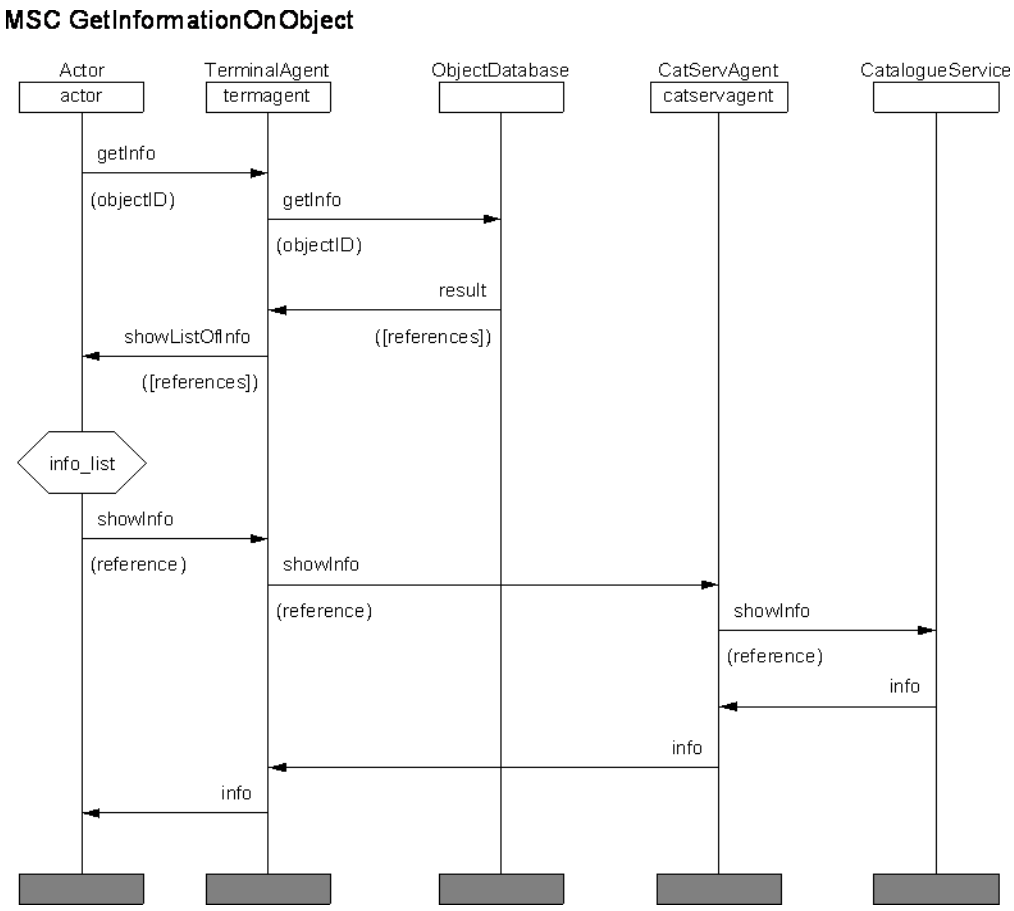


Figure 8–19: MSC for FA 6

To get information on an object, the actor inputs the ID for this object to the system. As described in the use case ‘Get information on objects’ in section 7.3.1.6, use case ‘search for objects and show position’ must have been performed first. The actor then already has a list of objects from which he can choose one object uniquely. An inquiry is made to the Object Database, which returns a list with references to all information belonging to this object. The list is presented to the actor, which then can choose the information he wants to view.

In the figure is has been shown the case where the desired information is stored in the Catalogue Service. This will most often be the case, but the information might potentially reside anywhere.

8.4.7 Requirement FA 7

In this section, the following requirement will be described:

- FA 7: An actor shall be able to trigger alarm

The sequence of actions will be as shown in figure 8–20.

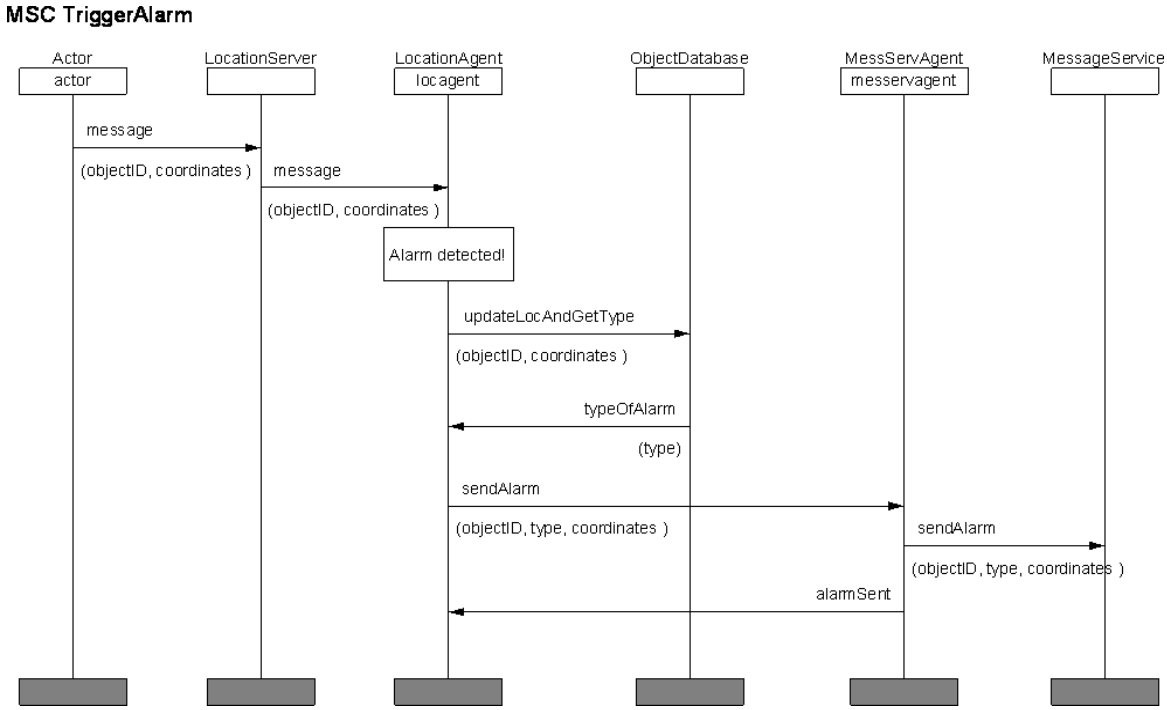


Figure 8–20: MSC for FA 7

Because this discussion has been based on functional requirements relating to actors, the figure shows what happens when an actor triggers alarm. The same thing happens however when any object triggers alarm, so therefore the term ‘object’ is used in the following.

As figure 8–20 shows, an object first sends a message to its LocationAgent. The message is just denoted ‘message’ here, because many different messages can potentially trigger alarm. It could for instance be a notify message telling that the object has been moved, or a message telling that someone has presses an alarm button on a tag. The LocationAgent for the object detects alarm for the message, and sends a request to the Object Database to update the database with the new location of the object, and to get some details on the triggered alarm. Because the interface to the Message Service is not specified yet, as mentioned before, it is not clear what form a message sent to it will take. In any case, some information needs to be retrieved from the Object Database and included in the message. In the figure, the type of the alarm is obtained, but this is just shown as an example and can be any number of parameters.

Finally, an alarm message is sent to the Message Service, which forwards it to the proper destination.

An alarm message is also sent to actors logged in to the SAPT with role 4. This is shown in figure 8–22 on page 80.

8.4.8 Requirement FA 8

In this section, the following requirement will be described:

- FA 8: An actor shall be able to monitor objects

The sequence of actions will be as shown in figure 8–21.

First, the actor submits parameters specifying which objects he wants to monitor. This can be to monitor certain objects, for instance all dialysis machines at the hospital, or to monitor a certain area, for instance the cancer department.
A search is performed in the Object Database to find all objects matching the parameters, and a list with these objects is sent back to the TerminalAgent. A message is then sent to the LocationAgent of all the objects that matched the search. In the figure only one LocationAgent is shown, but potentially very many LocationAgents will receive the message. If the object belonging to this LocationAgent is not currently tracked, a message to start tracking the object is sent to the Location Server. After this, new locations for the objects are returned to their LocationAgent, which forwards them to the TerminalAgent. The LocationAgents also updates the Object Database with the new locations. Upon receiving a new location for an object, the TerminalAgent sends a request to the Map System to get a map showing the new location of the object, along with the old locations for all the other objects.
This sequence of actions is repeated until the actor sends a message to the system telling that he wants to stop monitoring. If tracking was started on any objects solely for this session, messages are sent to the Location Server to stop tracking for these objects.

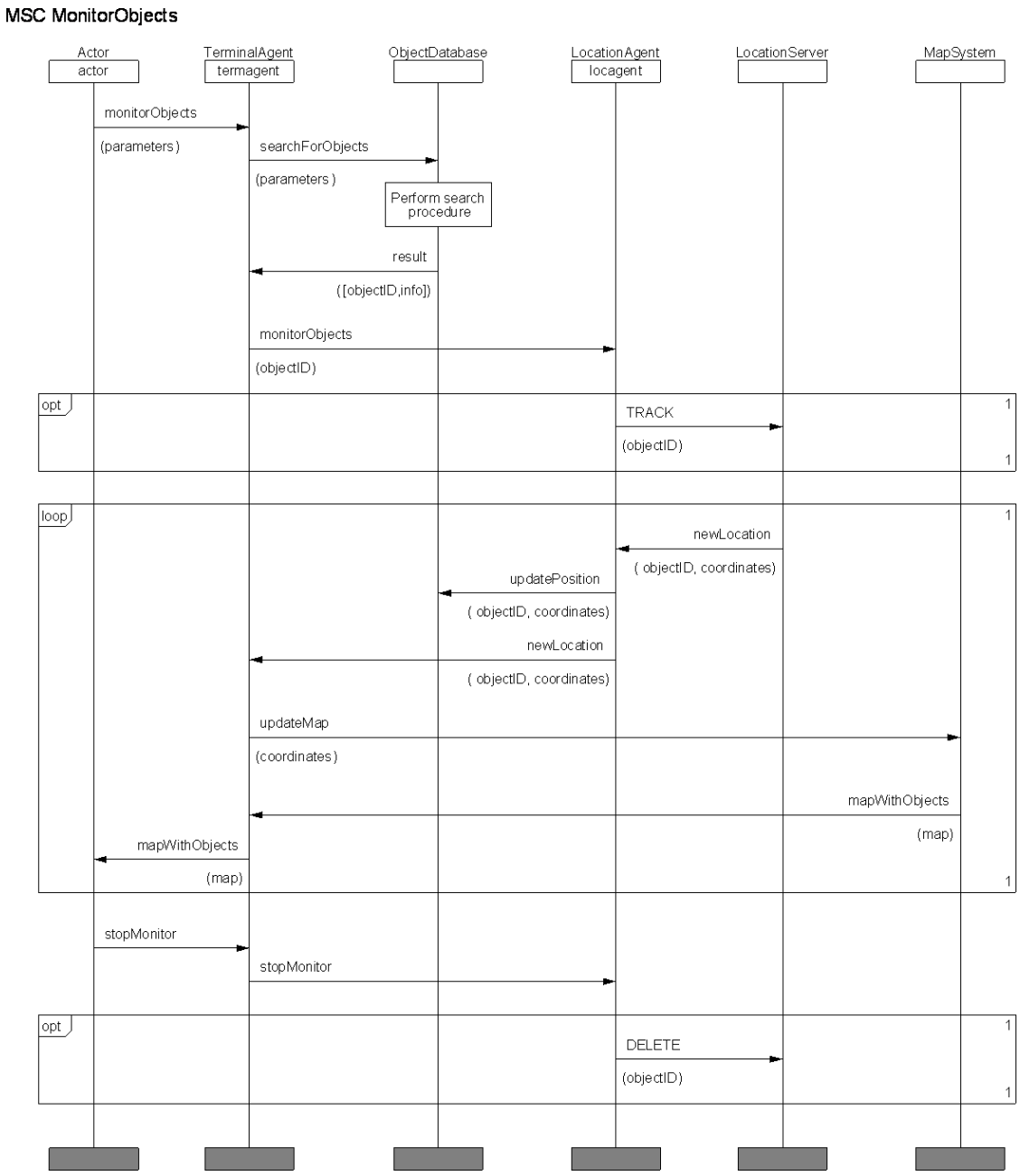


Figure 8–21: MSC for FA 8

8.4.9 Requirements FA 9 and FS 4

In this section, the following requirements will be described:

- FA 9: An actor shall be able to receive alarm
- FS 4: The system must be able to handle alarm messages from the Message Service

Figure 8–22 shows the case where an alarm is triggered in the SAPT itself, while figure 8–23 depicts the case where an alarm is triggered elsewhere at the hospital. This can for instance be a fire alarm that has gone off, a window that has been broken, etc.

The first case starts with the same sequence of actions as shown in figure 8–20, where an object triggers alarm in the system. The difference here is that the alarm message is sent to actors in the system. These will be actors logged in to the system with role 4. A routing table, for instance, can keep track of which TerminalAgent processes belongs to actors logged in to the system with role 4. The alarm message, and a map showing the location of the alarm, will be displayed on the actors' terminals until they acknowledge it. This can for instance be by pressing a button. In the figure, only one actor is shown, but the message might be sent to many actors.

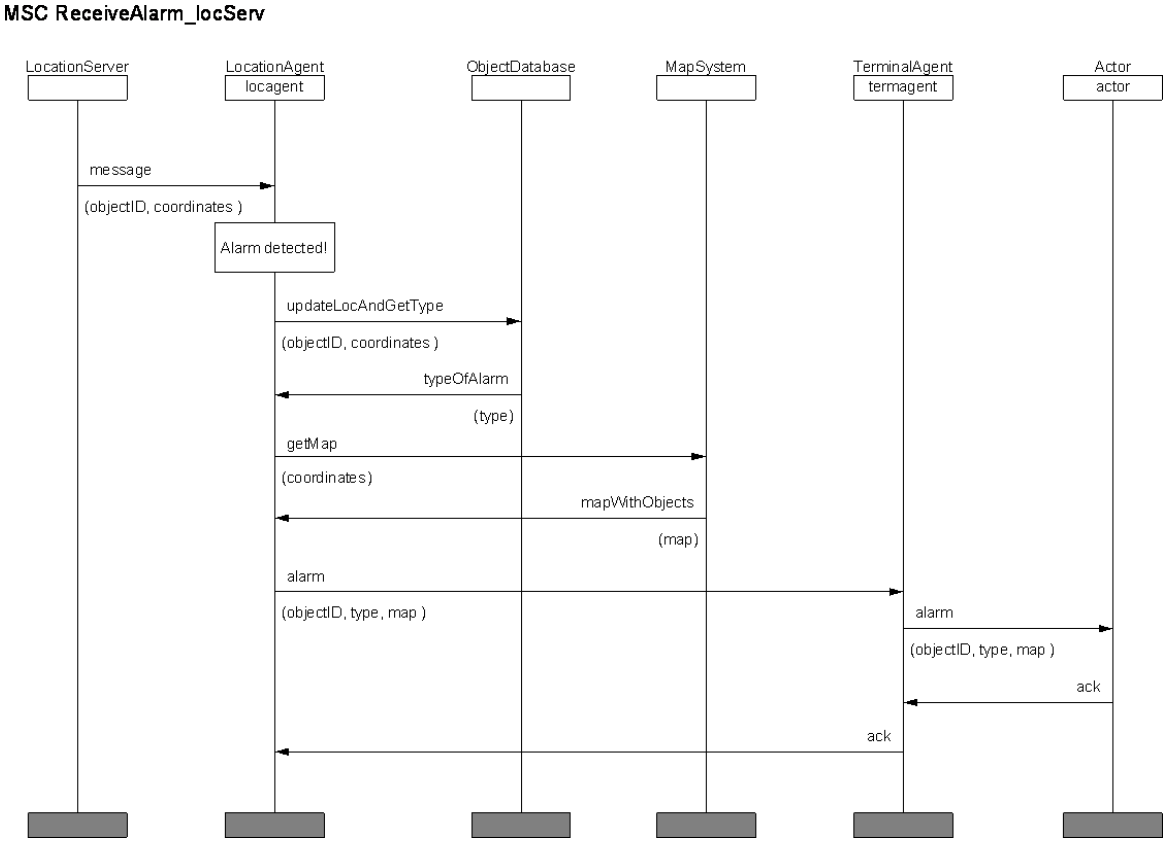


Figure 8–22: MSC for FA 9

The case where an alarm originates elsewhere at the hospital is shown in figure 8–23. When the MessServAgent detects that an alarm message has arrived, it creates a new instance of the process MessServSession. All further processing needed to handle the request is done by this instance. As shown in the figure, this includes getting an up-to-date location for the object, getting a map, and sending an alarm message to all actors logged in to the system with role 4. The instance is terminated after all actors have acknowledged the alarm message.

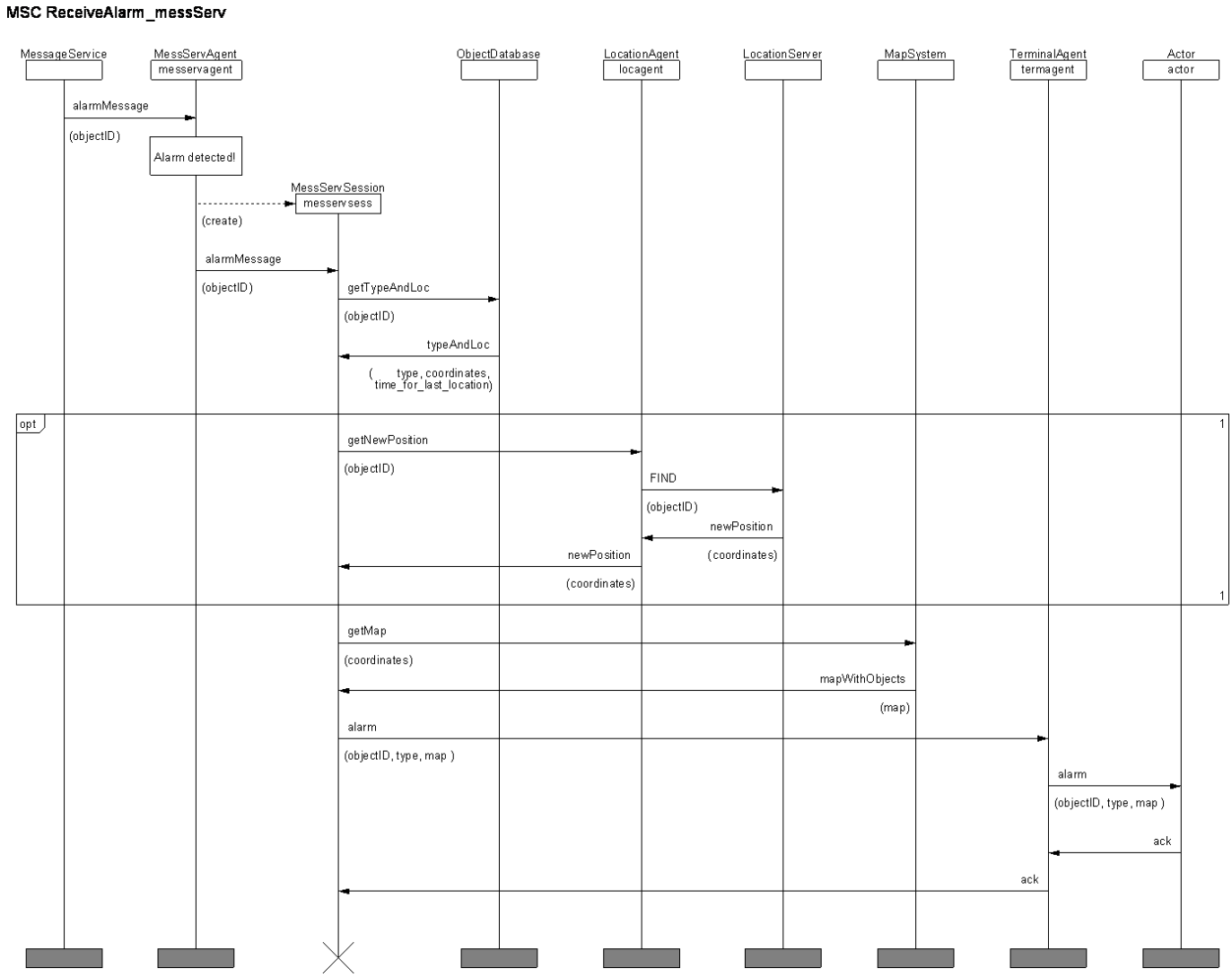


Figure 8–23: MSC for FS 4

9 Implementation design

In chapter 7, the requirements specification for the system was given, and chapter 8 presented the functional design. The next step in the development process is to design a physical system that will implement the specified functionality. This can be seen as mapping the abstract system defined in the functional design to a concrete system made up of hardware and software components. The goal of implementation design is to define this mapping and the concrete technical system. [32]
The concrete, physical environment for the system that is considered in this thesis is St. Olav's Hospital in Trondheim, Norway.

Because of limited amount of time available when working with this thesis, the implementation design and the implementation will not be performed thoroughly. It is recognized that the system will need to be distributed, components need to be replicated, etc., but subjects like this will not be studied at all. What will be in focus is the communication between the different parts of the system.
To summarize; the hardware design of the system is not considered at all, and the software design is only considered partly.

9.1 Choices regarding positioning

The first thing that was done in the implementation design process was to make a decision about which position techniques that would suit the system needs best. As the system is not to be build completely in this thesis, the decisions made here is to be viewed as guidelines for a future full implementation of the system. The solution presented is the choices that the author of this thesis thought was best, based on what was discovered in chapter 4: 'Techniques for positioning'.

Guidelines for which objects that need to be positioned are also made. They are based on what was found in section 7.3.6: 'Functional requirements relating to tags'.

9.1.1 Position techniques

As mentioned in section 4.6, the evolution of tags is very rapid, and new prototypes are presented quite often. Conversations with some of the providers of positioning equipment have

revealed that they are willing to build tags that will suit the needs at a hospital. Therefore, the capabilities and functionality of current tags has not been considered when selecting technology.

It has been found that a combination of GSM and Radionor positioning will suit the needs at St. Olav's Hospital best. The thought is that these two systems can work together, and both provide positions for objects to the Location Server. The functionality of the system is depicted in figure 9–1.

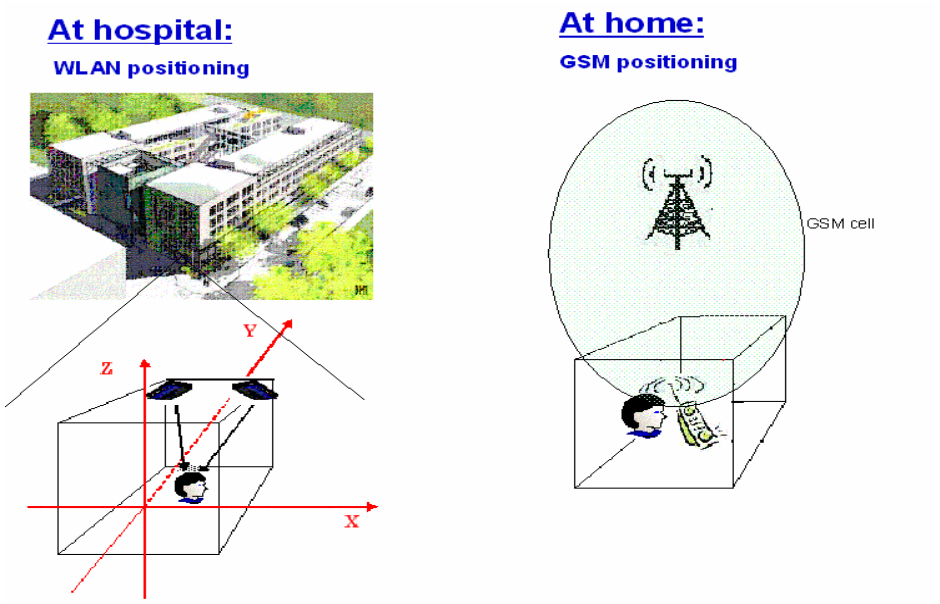


Figure 9–1: The solution: WLAN and GSM positioning combined

The technology from Radionor will be used inside the hospital to locate objects. The main arguments for choosing Radionor instead of Ekahau (WLAN), Sonitor (ultrasound) or the RFID technology are listed below.

- By using the RFID technology, the position of objects can only be obtained when passing interrogators. This has not been considered accurate enough.
- A complete WLAN infrastructure will be installed at the new St. Olav's Hospital, which will cover all indoor areas of the hospital. A lot of the infrastructure needed to use WLAN positioning will then be present, and this implies using WLAN instead of ultrasound.
- By using a WLAN technique, all objects that have a WLAN connection, for instance PDAs or laptops, can easily be included in the system without needing a separate tag. This also contradicts using ultrasound, as all objects will then need a tag.
- All beds for patients will have a terminal connected to it at the new St. Olav's Hospital. Patients can use these to watch films, surf the Internet, chat with friends, etc. If WLAN is used, beds can be included in the system without needing a separate tag.
- When choosing between Radionor and Ekahau, there had to be made a choice between a big hardware installation (install many Radionor RadioEyes), or a big software installation (install an Ekahau software client in all terminals that is to be positioned). It was considered more scalable to install RadioEyes in all the buildings. When the RadioEyes

are already installed, new terminals can be positioned without any new configuration. With Ekahau, each new device will need to install software. A considerable calibration procedure also needs to be performed when first installing it.

- A last advantage with Radionor, is that the company is located in the city of Trondheim, Norway, where also St. Olav's Hospital is. Through communication with employees at Radionor it has been given an understanding that many things can be done to adapt their product to suit the needs at the hospital. As Ekahau is situated in the US, it is harder to get a close communication with them.

GSM positioning will be used as a complement to Radionor, to make it possible to locate objects currently not at the hospital. The objects in mind here will be employees at the hospital. The accuracy will not be good, as explained before, but good enough to be able to determine if, for instance, a nurse is at home, shopping in the city, out travelling, etc. This can be of interest in an emergency, for example. It will be better to call a nurse that is home, than one that is away in another city.

9.1.2 Tags

This section will discuss how tags are to be used in the system.

9.1.2.1 Tags for equipment

All valuable equipment should carry a tag. This can be medical equipment, beds, TVs, monitors, etc. Whether *all* equipment should carry a tag is a question about money. One has to see the cost of adding a tag to equipment relative to the chance that somebody steals the equipment, or that the equipment is lost. Equipment that might not be critical to track includes chairs, tables, wardrobes, etc.
Equipment should have tags that emit signals when moved, but not when they are standing still. This way, there will not be excess messages that are transmitted in the system. It is not interesting to receive a message every 10th second telling that a bed is in the exact same place that it has been the last week.
Whether tags need to be pingable depends on whether all equipment shall be tracked. If this is the case, ping is not needed. Otherwise, if some equipment that is not tracked needs to be found, this function is needed.

9.1.2.2 Tags for visitors

In section 7.3.6.2: 'Tags for visitors', it was discussed whether visitors should carry tags at all. It has been decided that they need not to. It would take too much time and resources to give all visitors a tag. It was considered more cost-effective to restrict the access for visitors by using locked doors, for instance. In addition, since there will be tags on all valuable objects, for instance TVs, these will give alarm anyway if somebody tries to remove them.

9.1.2.3 Tags for patients

It has been found that not all patients need to be positioned, for the same reason as for visitors. There is only a need to track certain patients, such as mental patients or unstable patients. Other patients does not need to be tracked, but can have an alarm tag if they want to. This can for instance be older persons that need help when they have fallen, etc. The location of these persons can also be obtained by using the tag.

9.1.2.4 Tags for staff

When it comes to staff, all staff should carry a tag. This 'tag' can be a WLAN enabled device, an integrated alarm/positioning unit, or a separate tag just to be able to be positioned.
It should be possible to ping this tag, as there will not be feasible to track all staff at any time.

9.1.2.5 General

In general, tags that are to be used for the messages `TRACK`, `NOTIFY` and `NOTIFY_WHEN_MOVED` should emit signals every time they are moved, but not when they are lying still. In this way, they do not generate excess load on the WLAN infrastructure.
Tags used only for `ALARM` messages do not need to be pingable or emit signals when they are moved. They only need to send signals when an alarm is triggered, for instance if someone presses a button on them.
Tags used only for `FIND` messages, such as tags for staff, does not have to emit signals every time the person moves, as this would generate a large message load on the system. These tags should however be possible to ping.

9.2 Communication

To find an efficient way for communication between the SAPT and the different components it interfaces is one of the major tasks in this study.
A first thing that has to be discussed is the type of communication that is needed in the system.

According to [32], there are two different ways to communicate:

- by *signal units* of messages transferred as discrete information packages
- by *continuous signals* or values transferred continuously

Continuous signals might be read repeatedly, whereas signal units are normally read once.

It has been decided that to use signal units for all messages will suit the communication in the SAPT best. To communicate a signal unit, a unit-oriented media can be used. This can be implemented most directly by using message queues in software.

9.2.1 Synchronization

Synchronization is the act of aligning the operations of different concurrent processes in relation to each other.
There exist two different types of synchronization in communication according to [32] and [40]:

- *asynchronous communication*; which is a buffered communication scheme in which the sender may produce infinitely many signals without waiting for the receiver to consume them. Independent clocks are used at the transmitter and the receiver
- *synchronous communication*; in which the sending operation and the consuming operation occur at the same time. This is necessary if there is no buffer between the processes. Synchronised transmitter and receiver clocks are used

One can also classify the synchronization of interactions in the categories *time-dependent synchronization* and *time-independent synchronization*.

It has been found that a time-independent synchronization scheme with asynchronous communication fits the needs of the SAPT. In this case there is a buffer, normally a FIFO (First

In, First Out) input queue, between the interacting processes. The sender puts messages into the buffer, and the receiver removes the messages later. The buffer capacity determines how many messages the sender may produce ahead of the receiver.

Asynchronous message passing eases the communication between components, and suit the needs of many independent and parallel processes. It allows the involved parties to handle messages at their own speed, and the interacting objects may all continue currently. In the SAPT, both the actors at their terminals, the Location Server, The Message Service and the Catalogue Service may input messages to the system independently of each other.

9.2.2 Message sending

The most straightforward way for communication among software modules is by procedure calls. Procedure calls use a synchronous request-reply (sometimes referred to as 'call/wait') protocol which involves blocking of the sender until the receiver fulfils the request [41]. This means that the receiver process takes pre-emptive priority over the sender process.
Among the criteria for using this type of communication is that only one process may wait to receive messages form the environment, and that for each message sent in to the system, not more than one reply message is returned to the sender. Clearly, this is not the case for the system depicted in this thesis. Four different processes are receiving messages from the environment, and several reply messages might be returned to a sender. This is for instance the case when TRACK is set for an object.
Normal procedure calls are therefore not sufficient, and buffered communication is needed.

In the general case, activation must be separated from communication. To obtain this, the communicated messages must be passed through shared data records (buffers or queues), and activation through control programs that schedule the activation. This supports asynchronous communication between processes. Message queues provide temporary storage when the destination is busy. By using this, the receiver process need not have pre-emptive priority. Buffered communication with sufficient buffer capacity is therefore a very general and flexible solution.
The drawbacks with buffered communication are that it is slower, in most cases, than direct procedure calls, and that it is not supported by many programming languages. [32]

9.3 Concurrency and priority

Concurrent processes proceed independently of each other without any mutual ties apart from those imposed by explicit interaction. Several different processes run concurrently in the SAPT. With reference to figure 8–4 on page 60, a message might arrive at the CatServBlock from the Catalogue Service telling the system to delete an object at the same time that an alarm message from the Location Server is received at a LocServBlock.

Physical components in the real world truly behave in parallel. This imply that if one map each of the processes in the SAPT to a separate physical object, operations within different components will proceed in parallel to each other at the speed of the performing hardware. Inside the different components, however, there will not be a true parallel behaviour of received messages. For instance, the Router process might receive messages from both the CatServBlock and the TerminalBlock at the same time. Due to the nature of processes that run in hardware,

the actual processing in the Router will happen sequential. This means that the messages will be handled one at a time.

A decision can be made to run some of the processes on the same piece of hardware. The operation of these processes will then also be sequential and not parallel. The reasons for not dedicating a separate hardware resource to each of the processes can be many, but most specific is the cost of hardware.
As said in the introduction to this chapter, the hardware structure of the SAPT will not be considered in this thesis, so how the different processes are physically distributed will not be discussed any further.

That not all messages arriving at a process can be handled in true parallel, but rather sequential, has impact on the input queues of the different processes. As an example, an alarm message arriving at the system is more important than a request from an actor to get information on an object. To handle this, scheduling and priority mechanisms are needed.

This thesis will not go in detail of how scheduling and priority can be realized. Interested readers can consult [32], for instance. What will be discussed in the following are priority issues relating to the different blocks with processes. These issues will have to be considered when implementing the system.

9.3.1 TerminalBlock

The TerminalBlock for an actor will never receive any requests from the actor that will need special priority. The actor might request to ‘search for objects and show position of these’, or to ‘manage objects’. These are not considered critical requests. If the actor has to wait some extra time to get a response, this does not give any other consequences than that the actor might get angry.
If the actor is logged in to the system with role 2 or 3, the belonging TerminalAgent process will not receive any messages at all from the rest of the system. If an actor is logged in to the system with role 4, however, the TerminalAgent for this actor can receive alarm messages from the system. These messages can originate at either the LocServBlock or the MessServBlock, and they are considered important. The messages from the LocationAgent and the MessServAgent are considered of equal importance, but they are more important than messages originating at the actor. A way to realize this is that the TerminalAgent always checks in the input buffer from the Router first, before processing messages originating at the actor’s terminal.

9.3.2 CatServBlock

Since the CatServBlock does not handle any alarm messages, it has been found that no special priority mechanisms are needed in this block.

9.3.3 MessServBlock

The MessServBlock can receive two different messages from the environment; either a ‘search for objects and show position’ message, or an alarm. Clearly, the alarm message needs to be given priority. This can be realized by having two separate input queues for messages originating at the Message Service, one for alarms and one for other messages. The queue for alarms is given priority.

The MessServAgent will also receive alarms originating at the SAPT. These are to be sent to the environment, and are the only messages that it will receive from the system. The alarms from the *system* are considered more important than alarms from the *environment*, so they should be given priority. The reason for this is that alarms originating at the SAPT use the Message Service to send the alarms to the proper persons. Alarms coming from the environment are just notifications to those logged in with role 4 (safety responsible/administrator). Other actors in the environment have already received the alarm.

9.3.4 LocServBlock

As decided in section 8.3.3: 'LocServBlock', there will exist one LocationAgent process for each object in the system. The processes will receive many messages, both originating at the Location Server, and at the system.
It has been found that messages originating at the Location Server will be more urgent than those coming from elsewhere in the system, and should be given priority. The messages from the SAPT can be 'start tracking this object', 'find this object' etc. These can be handled by a normal input queue.
The current Location Server is only a prototype, as mentioned earlier. It has been recognized that the current operation of the server is not sufficient to realize needed priority mechanisms. The Location Server will need to support an `ALARM` message that can be sent from objects to the server, in addition to already existing messages (such as `FIND`, `TRACK`, etc.). The LocationAgents can then have one input queue for `ALARM` messages, and another for the other messages.
Something that needs to be evaluated is if there shall exist a third input queue, for the messages `NOTIFY` and `NOTIFY_WHEN_MOVED`. Both of these might trigger alarm, and can be considered more important that `FIND` and `TRACK` messages.

Priority also needs to be implemented in the Location Server. If alarm messages must wait to be sent from the Location Server, it does not really make a difference whether or not if the LocationAgents implements priority.

9.4 System start-up

When starting up the SAPT, a very large number of objects need to be added to the system. These objects can potentially be very different, and have different information connected to them. Tracking will have to be started for some of them, while others shall be set up to receive alarm.
It will not be feasible to add all of these objects manually one by one. Some mechanism has to be made that makes it easy to add more than 50 000 objects to the system.

10 Evaluation of a prototype made in 2003

In the summer of 2003, the author of this thesis worked at Bravida Geomatikk AS. The task for the summer was to develop a prototype that provided some basic functionality for an indoor positioning system. The goal was to give a 'proof of concept' that could be demonstrated to potential customers.
It has been decided to denote this implementation 'prototype', to separate it from the demonstrator made specially for this thesis, as described in section 11: 'Implementation'.
The prototype uses indoor maps from the 'Telenor-bygget' at Tyholt.

10.1 Functionality

The functionality of the prototype corresponds to that described in this thesis for actors logged in to the system with role 2.
The main part of the prototype is designed as a set of JSP pages (see [42]), which are accessible from the Internet. It demonstrates the functional requirements FA 1: 'Log in', FA 5: 'Search for objects and show position', FA 6: 'Get information on objects' and FA 10: 'Log out'.
There was also made a separate Java program that demonstrates the functional requirement FA 4: 'Be positioned'.

10.1.1 JSP system

This section will describe the functionality of the JSP system.
The JSP pages that were made for the prototype can be seen in appendix C: 'JSP pages from prototype'. It is recommended that the reader study these before reading on, as it will make the following discussion easier to understand.

An actor can first log in to the system with username and password. A search page is shown right after this, along with a map representing the first floor at Telenor-bygget. The page makes it possible to search for objects matching certain parameters. The actor can select the parameters he wants to, and press a 'search' button. After this, a list is shown, containing all objects matching the search. The actor can then select to view the location for some or all of the listed

objects. The reason for doing it this way is that the search may potentially return very many objects. It could be confusing to show all of them on a map at the same time.
When the actor has selected the desired objects, he can press a 'show position' button, and a map showing the objects is displayed. A button-row with four buttons on the right hand side shows that there are four floors in the building. If a button is clickable, it indicates that there are beds on that floor. If the actor places the mouse over the objects on the map, he can see which objects they are, for instance 'bed1', 'bed 5', etc.
On the left side of the page, a list of the objects shown in the map is given. The actor can select one of the objects uniquely, by marking a radio button next to it. If he then presses a 'show info' button, a list of links to available information for this object is given. When one of the links is pressed, the selected information is shown in a separate window. On the same page where the actor could press a 'show info' button there is also a button to 'reserve equipment'. This feature is not implemented, and during the work with this thesis it has been recognized that it might not be needed. The actor can rather do an 'update object parameters' request.
The JSP system communicates with the Object Database and the Map System.

10.1.2 Java program

A separate Java program was made to handle the communication with the Location Server. It receives new locations from the Location Server, and inserts them into the Object Database. This program has no user interface, and has to be started from a console. The program is set up to obtain positions for one object. If more than one object is to be found, one instance of this program will need to be run for each of the objects.

The Location Server used was a prototype version made at Telenor FoU in Trondheim. At that time, it supported the messages '`TRACK`' and '`FIND`'. The server could be set up either to return random 'dummy' locations for testing, or to return real location data.
To test the prototype with real locations, a test system that had been set up in the Telenor-bygget by Telenor FoU was employed. The system used technology from Radionor (see chapter 4.3.2: 'WLAN positioning with Radionor'), and one RadioEye was set up to collect locations. The RadioEye was also only a prototype. No tags were available, and only WLAN-enabled devices could be positioned, such as laptops and PDA's. A laptop with a WLAN card was used for testing, and when the Java program was started, positions for it was successfully obtained.
The Java program communicates with the Location Server and the Object Database.

10.2 Evaluation

This section will evaluate different parts of the prototype. It will discuss matters that could have been done better, and state obvious lackings.

The section will only study the functionality that has been made by the author of this thesis. It will not discuss shortcomings of other systems, for instance the Map Server, etc. This is discusses in chapter 12: 'Future work'.

10.2.1 User interface

The user interface of the prototype is consistent with what has been designed in chapter 7.5: 'Prototypes for the user interface'.

The path in the system; that the actor first performs a search, then selects some of the objects from the search to show on a map, and then can get information about objects, is also good. The map has been given quite a bit of space on the screen, to be as useful as possible.
The design of the user interface and the functionality there is probably one of the better parts of the prototype.

A disadvantage is that the user interface is only adapted to PC's and not to PDA's or other terminals.

10.2.2 Search procedure

The system uses a search procedure to search the Object Database for objects matching certain parameters. The search returns the objects with their location and a little information. The parameters are submitted by the actor from a JSP page.

The search procedure is not optimal at all, and takes a long time to perform. This can probably be greatly improved by a more experienced programmer. The main goal when developing the prototype was to make it work, and performance was not given a high priority. The search procedure is the single most time consuming part of the prototype.

10.2.3 Communication

The communication pattern between the different systems is shown in figure 10–1.

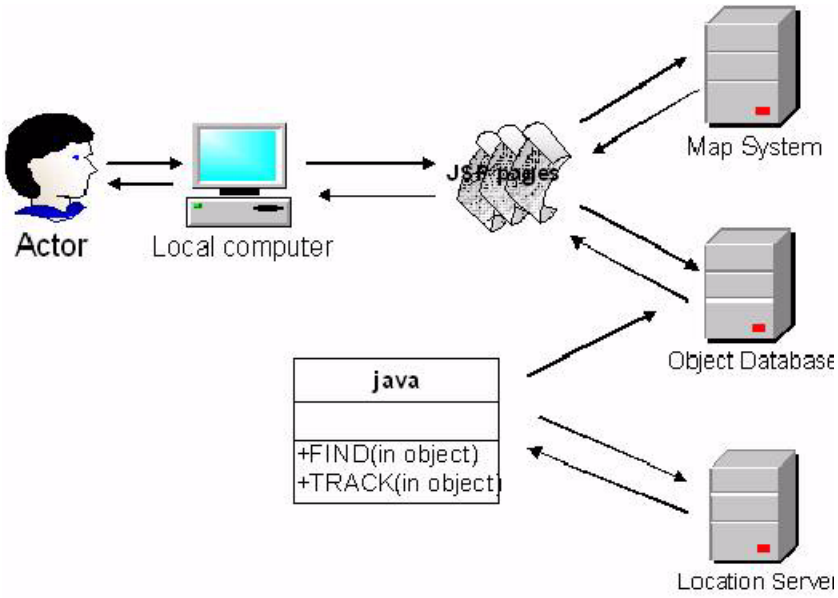


Figure 10–1: Communication pattern in prototype

The communication between components does not have a good design, and is probably not suited for the system that has been described in this study. As depicted in the figure, the actor interacts with the system through JSP pages. All communication with the Object Database and the Map System are handled as procedure calls from the JSP system. The Java program receives locations from the Location Server, and inserts the new locations in to the Object Database. The

JSP system and the Java program does not communicate in any other way than that the Java program updates the Object Database, and the JSP system fetches locations from the database.

It is not easy to include direct communication between components in this system. It is for instance not possible for an actor to detect an alarm message for an object originating at the Location Server in any other way than if the actor by chance asks the Object Database if an alarm has been triggered for the object.

10.2.4 Location updates

The prototype is constructed in such a way that for each object that is to be tracked, there has to be a separate Java program running for it.
The program starts tracking the desired object by sending a `TRACK` message to the Location Server. It then listens for responses containing new locations, which it inserts in to the Object Database. This works fine when there are a small number of objects in the system, but will not scale very well when there are 50 000 objects and 50 000 different programs.

That the only way an actor can obtain new locations of an object he is viewing is to send a new request to Object Database, is not very optimal either. Or- for actors logged in to the system with role type 2, this is the intended behaviour of the system. For actors with role 4 however, this is clearly not enough. When they sit at their terminal and monitor objects, they need to be able to receive new location information for the objects frequently, without having to ask for them repeatedly.

10.3 Summary

The prototype was made with one thing in mind: that it should provide specific functions outlined by the employer. No analysis of requirements, roles, functional design, etc. was made prior to the implementation, and solutions to problems experienced along the way were solved ad-hoc. The prototype was not made to work in a large scale, and was only suited for special situations.

11 Implementation

We now have a functional design and an implementation design, which are both inputs to the actual implementation.
All the main parts of the functional design have been made, but as mentioned in previous chapters, the implementation design has not made in detail. How the system is to be realized in hardware has not been evaluated at all, and only certain aspects of the software have been considered. The focus has been around the communication and concurrency aspects, and a detailed introduction was given to this.

This chapter will describe in detail an implementation of a demonstrator for the SAPT. The demonstrator is from now denoted 'SAPTdemo'.

11.1 Development environment

The SAPTdemo has been developed at Bravida Geomatikk's facilities at Tyholt in Trondheim. The reason for this is that the development required access to the Object Database, the Map System and the Location Server. This was available at Bravida Geomatikk.
The SAPTdemo was developed on a Microsoft Windows machine.

The source code for the SAPTdemo is written in Java [43]. The reason for choosing Java is that this was the programming language most familiar to the author. Java 2 Platform, Standard Edition (J2SE) version 1.3.1 was installed on the machine.
The code was written using the Eclipse tool. The Eclipse platform is designed for building integrated development environments (IDEs) that can be used to create many different applications, such as Java programs. [44]

11.2 JavaFrame

In section 9.2: 'Communication' it was found that a time-independent synchronization scheme with asynchronous communication would suit the SAPT best. It was also said that this scheme is not supported by many programming languages.

JavaFrame, developed by Ericsson [45], is a framework that supports this kind of communication. It provides a thread-safe environment, so one does not have to worry about threads when there are many concurrent processes running.

JavaFrame is a Modelling Development Kit that aims at improving the dependability of large, complex real-time systems implemented in Java. The framework provides classes of well-proven modelling concepts. By using these instead of just programming in plain Java, the abstraction level is raised.
JavaFrame can be viewed in two different ways: as an advanced API or as a specialised language with support for modelling concepts.

JavaFrame is built upon the experience that large, complex real time systems are best modelled in terms of active objects that interact asynchronously through messages. The behaviour of an active object is therefore described either by a state machine or by a structure of interacting state machines.
A JavaFrame system consists of a number of connected active objects. An important characteristic of active objects is that they may also be a composite, since an active object can have a substructure of active objects. Finally, a composite will contain a state machine. (For readers unfamiliar with state machines, see for instance [32].) All communication between active objects is done through mediators. This is shown in figure 11–1.

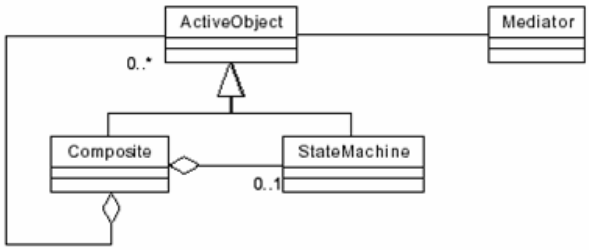


Figure 11–1: JavaFrame structure

A JavaFrame system is a system where active objects communicate asynchronously, and all active objects can be considered state machines. A state machine behaves in a reactive way. This means that when a message is consumed from the message queue of a state machine, a transition is performed. The communication is asynchronous because each state machine has its own message queue where incoming messages are saved. In this way, the relative speed of the different state machines do not matter.

The thought is that all state machines shall execute in parallel. This means in principle that state machines can be distributed on several processors. In JavaFrame, there exists schedulers that organize the execution of a set of state machines. A scheduler is associated with a Java thread, and the programmer is not to spin of any other threads in the JavaFrame system. A state machine must be included in the execution list of a scheduler to be able to be executed.

The schedulers are responsible for the execution of the state machines through their associations with Java threads, but conceptually it is the state machines that perform the transitions. This is an important point to remember for the programmer.

The state machines in JavaFrame are re-entrant, which means that many state machines refer to the same state machine structure. The states are separate objects with the transitions implemented in a method with the triggering message as a parameter. The last statement in a method will bring the state machine to a new state. This is shown in figure 11–2.

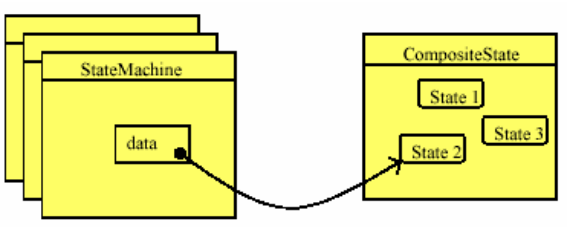


Figure 11–2: State machines that use the same behaviour description

The JavaFrame state machines do also have composite states. Composite states are states that have internal states. When entering a composite state, the model may specify an entry point that will define what inner state will be reached. A Composite State may be exited through exit points to reach states on enclosing levels.

Each state machine has an explicit queue of messages. These queues are managed by the scheduler. The scheduling regime is such that when a state machine has consumed a number of messages, the scheduler continues to the next state machine in a round-robin fashion. Thus within one scheduler, the input queues are fair.

Mediators are objects that represent the communication interface between active objects. Each state machine will address only the innermost mediators. These mediators are in turn associated with other mediators of composites to form the system architecture. The simplest mediators contain only a reference to another mediator, while more advanced mediators include routing, or even message translation.
Mediators minimize the need for the designer of one part of the system to have knowledge of completely different parts of the system.
[46]

As a summary, what JavaFrame does is to realize asynchronous communication, and make it possible to implement state machines directly. It also creates a thread-safe environment, so threads do not have to be considered when programming.

11.3 Implementation overview

Because of limited time when writing this thesis, an implementation of the full system was not feasible. A selection of what to implement had to be made.

After some considerations, a decision was made based on the following criteria:

- It was considered important to show that all components could communicate with each other in the depicted way, i.e. by means of asynchronous messages.
- It was decided to implement a little bit of a lot, instead of a lot of a little bit. This to be able to show some important principles, rather that to, for instance, implement the search procedure in a perfect way.
- Since a prototype showing some principles had already been made in 2003, the new demonstrator should emphasize other principles. I.e., since the previous demonstrator had a web interface, the new demonstrator is a Java application, etc.

11.3.1 Behaviour

The SAPTdemo is implemented as a Java application. This means that it is suited for actors logged in to the system with role 4, i.e. the safety responsible/administrator role. As previously described, this role has extended privileges compared to other actors in the system. It is for instance the only role in the SAPT that can receive alarms.

An activity diagram for the SAPTdemo is shown in figure 11–3.

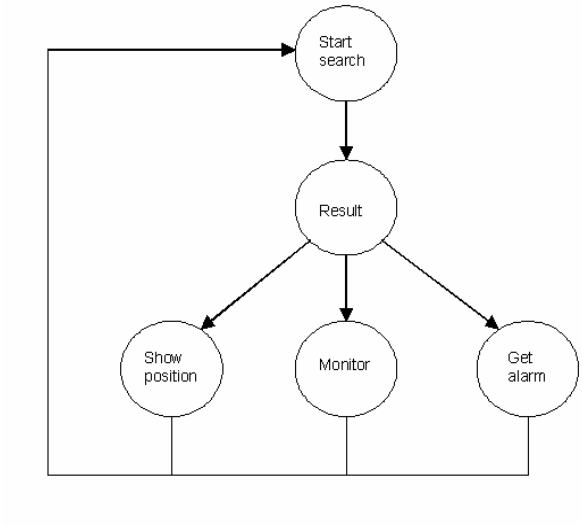


Figure 11–3: Activity diagram for SAPTdemo

An actor is first presented a view that makes it possible to search for objects matching certain parameters. The parameters are specified by the actor. After the search, the actor is presented a list of objects that matched the search, the ‘result’ view. From the list, the actor can select one of the objects uniquely. He can then either view the position of the object, start to monitor the object, or to start receiving alarms for the object. When the actor is done with this, he can return to the ‘search’ view and start the procedure all over again.

By means of the SAPTdemo, an actor can perform the following use cases:

- *FA 3: Update object parameters*. The actor can start monitoring an object, or start alarm for an object. This is typical examples of parameters that can be updated for an object.
- *FA 4: Be positioned.* A Location Server from Telenor FoU, which supports different position technologies, is used in the SAPTdemo. Both the technologies from Radionor and Ekahau are installed in the Telenor-bygget, and these can be used to obtain real locations for an object.
- *FA 5: Search for objects and show position of these.* The actor can search for objects matching certain parameters. After the search, he can choose to view the location of these.
- *FA 7: Trigger alarm*. See the explanation from *FA 4*.
- *FA 8: Monitor objects*. The actor can use the Java application to see the location of an object as it moves around. New locations are transmitted to the actor at regular intervals.
- *FA 9: Receive alarm.* The actor can use the Java application to receive alarms for an object. In the SAPTdemo, alarm is triggered if an object is moved out of a certain area.

A thing to note is that none of these use cases are implemented fully in accordance with what was shown in the MSCs in section 8.4: 'Process interaction'. This is because some simplifications have been made in the SAPTdemo.

11.3.2 Architecture

An overview of the implementation architecture is given in figure 11–4.

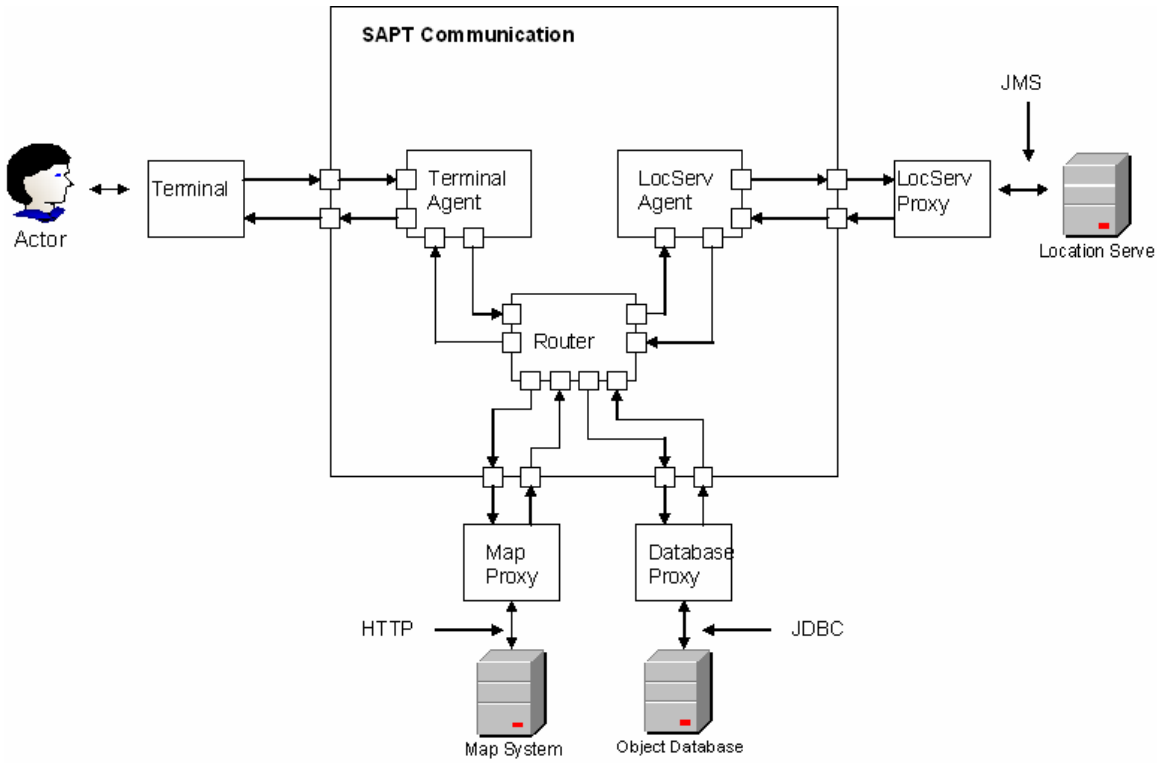


Figure 11–4: Implementation overview

The notation used inside ‘SAPT Communication’ is JavaFrame notation. The big squares denote different processes in the system. The small squares on all the edges denote input- and output mediators.
The boxes named ‘Terminal’, ‘LocServProxy’, ‘MapProxy’ and ‘DatabaseProxy’ are separate Java programs. ‘Terminal’ provides the graphical user interface (GUI) of the system, and is the component that makes it possible for actors to communicate with the SAPT.

In the SAPTdemo, all components are constructed at system start-up, and there will only exist one instance of the different components at any time.

11.3.2.1 The proxies

An explanation is needed on the proxies in figure 11–4.
Their main task is to translate between the JavaFrame system and other systems. As an example, the MapProxy translates between the JavaFrame system and the protocol that the Map System uses. They also work as a link between asynchronous and synchronous communication. Asynchronous communication is used inside the ‘SAPT Communication’ and between the ‘SAPT Communication’ and the proxies, while synchronous communication is used between the proxies and their respective components.

The external systems usually employ synchronous protocols, which makes it possible to let the proxies be stateless. This will however involve that a proxy is to be considered busy from the point that it receives a request, until it has responded to it. This is not a desirable feature if there are many actors in the system. One way to solve this problem is to let there be more than one instance of each proxy, for instance a ‘pool’ of proxies, or to create one instance of the proxy for each session that an actor starts.

The proxies create an abstraction layer above the external components, i.e. the Location Server, the Map System and the Object Database. If one for instance needs to connect to another Object Database (an updated version for example), the only thing that needs to be changed in the system is the DatabaseProxy. Besides, this component can use a proprietary protocol when communicating with the Object Database, without this creating problems towards the rest of the system.

11.3.3 Shortcomings

As mentioned earlier, the SAPTdemo is not fully in accordance with the designed SAPT. The reason for this is limited time available when writing this thesis.
The remaining elements can however quite easy be added to the SAPTdemo, since the fundamental framework and functionality have already been implemented.

The main shortcomings in the SAPTdemo are:

- It is only possible to show objects on the first floor, and the search procedure only returns objects currently on the first floor.
- It is only possible to show the location of one object at a time.
- It is only possible to monitor or to set alarm for one object at a time.
- It is suited for one actor only.
- No priority mechanisms are implemented in the processes.
- The Object Database is not updated with new locations from the Location Server.

- When monitoring an object, the object does not move across a fixed background map, but a new map is loaded each time a new location for the object is received.
- Non-functional requirements are not considered.

11.4 Graphical user interface

The user interface has been designed in accordance with the guidelines from section 7.5.3: 'Example of Java application interface'. Some simplifications have been made, but the main principles have been followed.
Technically, it has been created by making an instance of the Java class `Box`, and then adding two more `Box`'es inside it: one for the left side (the 'text side'), and one for the right side (the 'map side'). These two `Box`'es are then changed by adding and deleting different Java `JComponents` to them. The `JComponent` class provides support for creating text-fields, images, radio buttons, etc., that can be added to the application.
These classes are described in detail in the Java API. [47]

The user interface for the SAPTdemo can be seen in appendix F: 'System test'.

11.5 Connection to other components

This section will describe how the connection to the Map System, the Object Database and the Location Server has been realized in the SAPTdemo.
Java classes referred to in the explanation can be found in [47].

11.5.1 Connection to the Map System

The Map System is reached by a HTTP call from the Map Proxy.
First, one sets up a connection to the server where the Map System resides, using the `HttpURLConnection` class. This can be done with the following command:

```
HttpURLConnection connection = (HttpURLConnection)queryURL.openConnection();
```

The `queryURL` parameter is the HTTP request sent to the Map System. It will be on the form:

```
http://<location of the Map System>
?CX=10.438&CY=63.422&IW=550&IH=550&SC=500&<other parameters>
```

In this example, the x- and y-coordinates for an object is given, along with the desired width, height and scale of the map image.
The response is received as an `InputStream` at the terminal application.

To be able to use the HTTP response as an image in the Java application, some processing of the input needs to be done. In the SAPTdemo, it was decided to create an `ImageIcon` of the received `InputStream`. This was done because an `ImageIcon` is a component that can easily be included in a GUI that uses `JComponents`, as described in section 11.4. It is the `ImageIcon` that is sent back to the actor of the terminal application through the JavaFrame structure.

11.5.2 Connection to the Object Database

The Object Database is reached through a JDBC call from the Database Proxy. JDBC, short for Java Database Connectivity, is a Java API that enables Java programs to execute SQL statements. This allows Java programs to interact with any SQL-compliant database. See [48] for more information on JDBC.
The establishment of a connection to the Object Database can be performed by the following command:

```
Connection conn = Database.getConnection(url, "username", "password");
```

The `url` parameter specifies which database to connect to, and it will be on the form:

```
url = "jdbc:oracle:thin:@10.254.10.142:1521:brapt";
```

The url shown here specifies that JDBC will be used to establish a connection to an oracle server named 10.254.10.142 on port 1521.

The rest of the communication with the Object Database is done by calling methods from a proprietary Java API made at Bravida Geomatikk AS. The API needs to be installed on the machine where the SAPTdemo is to be run from, and can be included in the code by adding the line:

```
import no.geomatikk.database.*;
```

11.5.3 Connection to the Location Server

The Location Server used in the SAPTdemo is a prototype made at Telenor FoU in Trondheim. It is reached by a JMS call done by the LocServProxy. JMS is short for Java Message Service, and is an API that allows application components based on the Java 2 Platform, Enterprise Edition (J2EE) to create, send, receive, and read messages. For more information on JMS, consult [49].

To use JMS in the application, the following JAR files were downloaded from [50], and installed on the machine where the SAPTdemo was to be run from:

- `exolabcore-0.3.4.jar`
- `jms_1.0.2a.jar`
- `openjms-client-0.7.4.jar`

These APIs uses JNDI (Java Naming and Directory Interface) to store connection factories, topics and queues. In order to start using JMS messages, a JNDI `Context` needs to be created. This can be done by the following lines:

```
Hashtable props = new Hashtable();
props.put(Context.PROVIDER_URL, "tcp://129.241.219.152:3035/");
props.put(Context.INITIAL_CONTEXT_FACTORY,
     "org.exolab.jms.jndi.mipc.IpcJndiInitialContextFactory");
Context context = new InitialContext(props);
```

The `Context.PROVIDER_URL` specifies the `URL` of the Location Server. As the `URL` shown here depicts, the protocol used for communication is TCP.
When the `Context` is established, one can start sending messages between the client and the Location Server. This is done by using methods from the downloaded JAR files.

11.6 Provided method interface

This section will describe the interface that the SAPTdemo provides to actors of the system. It describes the messages that actors can send to the system, and the returned responses.
All messages are sent via the JavaFrame structure, i.e. the structure shown inside the 'SAPT Communication' on figure 11–4 on page 99.

The source code written for the demonstrator is cited in appendix G: 'Source code', and on the enclosed CD. The SDL process graphs of the SAPTdemo made prior to the implementation are shown in appendix D: 'Process graphs'.

11.6.1 SearchForObjectsM

This is a message that makes it possible for an actor to search for objects matching certain parameters. The message is sent from the actor's terminal to the DatabaseProxy.

method declaration:
`SearchForObjectsM(String type, String avd, String status, String navn, String unikId)`

- **type** the type of the object
- **avd** the ward that the object belongs to
- **status** the current status of the object
- **navn** used when searching for persons. The name of the desired person
- **unikId** used when searching for one specific object. The unique identification number for the object

return `ResultListM(ArrayList result)`
A message from the DatabaseProxy providing a list of objects matching the search. The list contains the object ID, a short description and the current coordinates for all the objects. The DatabaseProxy has obtained this information from the Object Database.

11.6.2 ShowMapM

This message is sent as a request to view a map. A point, denoted by a longitude and latitude, is marked with a pin on the returned map. This message is sent from the actor's terminal to the MapProxy.

method declaration: `ShowMapM(String xcoord, String ycoord)`

- **xcoord** the longitude of the point
- **ycoord** the latitude of the point

return `MapWithObjectsM(ImageIcon map)`
A message from the MapProxy containing an image of the desired map. The image is retrieved from the Map System.

11.6.3 MonitorObjectsM

This message is sent when an actor wants to start monitoring an object. It is sent from the actor's terminal to the LocServProxy.
In this context, monitoring an object means receiving updated location information for the object at regular intervals. The LocServProxy gets the location information from the Location Server.

method declaration: `MonitorObjectsM(String unikID)`

- **unikID** the unique identification number for the object

return `MapWithObjectsM(ImageIcon map)`
Upon receiving new locations for the object from the Location Server, the LocServProxy sends these back towards the actor's terminal in messages called `NewLocationM(String xcoord, String ycoord)`. As the messages reach the TerminalAgent, the TerminalAgent sends a message to the MapProxy asking to get a map where the point denoted by `xcoord` and `ycoord` is marked with a pin. The messages with new location information are therefore never returned to the actor's terminal. The actor only receives new `MapWithObjects` -messages. He keeps on receiving these messages until he sends an `EndMonitorM`.

11.6.4 EndMonitorM

This is a message that an actor can send to stop monitoring an object, i.e. stop receiving new location information for an object. The message is sent from the actor's terminal to the LocServProxy, which forwards the request to the Location Server. No response is returned to the actor when sending this message.

method declaration: `EndMonitorM()`

11.6.5 StartAlarmM

This message is sent when an actor wants to start receiving alarms for an object. It is sent from the actor's terminal to the LocServProxy. The LocServProxy forwards the request to the Location Server.
In this context, receiving alarm for an object means that the actor receives a notification if the object is moved outside a legal area. In the SAPTdemo, alarm is given if the object is moved more than 25 meters away from a point with longitude 10.4379575192261 and latitude 63.422366770019496. An alarm message from the Location Server is only sent once, i.e. the first time the object is moved out of the area.
Upon sending a `StartAlarm` -message, the actor gets two return messages back.

method declaration: `StartAlarmM(String unikID)`

- **unikID** the unique identification number for the object

return `AlarmToTerminalM()`
return `MapWithObjectsM(ImageIcon map)`
When the LocServProxy receives a notification from the Location Server that an alarm has been triggered for an object, it sends an `Alarm` message to the TerminalAgent. This message contains

the longitude and latitude for the location of the alarm. Upon receiving this message, the TerminalAgent sends an `AlarmToTerminal` message directly to the terminal, which then displays an 'ALARM!' message to the actor. The TerminalAgent also sends a message to the MapProxy asking to get a map where the point denoted by the longitude and latitude is marked with a pin. The image is returned to the actor's terminal, showing the location of the alarm.

11.6.6 EndAlarmM

This is a message that an actor can send to tell the system that he does not want to receive alarm messages for an object any more. The message is sent from the actor's terminal to the LocServProxy, which forwards the request to the Location Server. No response is returned to the actor upon sending this message.

method declaration: `EndAlarmM()`

11.7 Testing

Detailed tests have been performed on the SAPTdemo to ensure correct operation. The tests were performed on a Microsoft Windows machine at Bravida Geomatikk's facilities at Tyholt in Trondheim.

One specific test, performed after the development had finished, is described in appendix F: 'System test'. Based on this test, it can be concluded that the demonstrator worked as intended.

> **NB!** For a slide show of the logical operation of the SAPTdemo please take a look at the file *systemTest.pps* on the enclosed CD, or visit this web-address:
> http://www.stud.ntnu.no/~sjovoll/systemTest.pps.

A thing to note is that all development and testing was done with the Location Server in test-mode. This means that it did not obtain actual location information by using a location technique, for instance WLAN positioning. It just returned randomly generated positions for objects. The random positions were such that they were located inside the Telenor-bygget at Tyholt.
The reason why the SAPTdemo has not been tested with the Location Server in a 'real' mode is that the author of this thesis did not have access to a laptop with a WLAN card. To receive real location information from the Cordis RadioEye that has been set up at Telenor-bygget, this is required. This test was however not considered critical, as it had been tried successfully during the summer project that the author worked on in 2003. Staff at Telenor FoU had also performed tests involving the Location Server and the Cordis RadioEye recently, so it was concluded that it would not be a problem to obtain real location information.

11.8 Summary of implementation

One of the tasks for the SAPTdemo was to show how the communication between the different components could be realized. This has proven successful. An actor at a terminal can communicate with the Object Database, the Map System and the Location Server by means of time-independent asynchronous messages. The demonstrator has also shown the basic concepts behind the functional requirements FA 3, FA 4, FA 5, FA 7, FA 8 and FA 9.

The SAPTdemo works as intended, but is a bit slow. The most time consuming parts are actually the call to the Object Database to search for objects, and the call to the Map System to get maps. The sending of messages through the JavaFrame structure is fast, and does not add any significant time to the processing of requests.

With regards to the source code written, the hardest part was actually to make the graphical user interface (the GUI) look good, and to get it updated at the correct instances. To implement the communication in JavaFrame was not that difficult, once the process graphs were designed (shown in appendix D: 'Process graphs'). In addition, Java threads did not need to be handled, which was a great relief.

Some of the source code in the demonstrator is reused code from the prototype implemented in 2003. As an example, the search procedure used to find objects in the Object Database is reused with only small modifications. The call to communicate with the Location Server is another example of reused code.
According to [32], the term 'reuse' means 'use again' or 'use more than once'. Reuse is intended to minimise the effort spent in designing software. The main benefit of reuse is that one does not have to 'reinvent the wheel repeatedly'.
The author of this thesis spent about a week developing the search procedure last summer, so it was a great relief to be able to reuse this component when implementing the SAPTdemo.

12 Future work

As the requirements specification and the design of the system give an impression of, the implementation of the SAPT is a big and comprehensive project. For the SAPT to be employed commercially, there is still a lot that has to be done. Because of limited time available, only a demonstrator has been implemented, and not the full system. The Catalogue Service and the Message Service do not yet exist either, so it is quite impossible to build the total system at this moment.

This chapter will describe improvements the author has discovered to be needed during the work with this thesis. Only existing components will be mentioned.

12.1 General

The functional requirements and the functional design are about finished for the system, but other areas are not completed. An area that has been left quite open is the non-functional requirements, which will need to be emphasized when implementing the full system.

The SAPTdemo is just a small demonstrator showing some important principles, rather that a tentative on a real implementation. It is therefore a lot of implementation work left for the system. As a basis for a future full implementation of the system, it would be natural to make use of the functional design from this thesis.

12.2 Location Server

Suggestions for improvements in the current Location Server are:

- Construct a new message type in addition to the current `FIND`, `TRACK`, etc., called `ALARM`. This to be able to separate high-priority alarm messages from other message types.
- Implement priority mechanisms in the server, so that important messages are processed first.

A question that needs to be discussed is how much logic that is needed in the Location Server. Currently, a lot of information needs to be replicated in the Location Server and the Object Database. For instance, if an object is to give alarm when moved outside the cancer department, the coordinates for this area have to be stored in the Location Server.
However, to be able to search for 'all objects at the cancer department', the coordinates for the cancer department need to be stored in the Object Database too.

One solution to this is to have a less intelligent Location Server that only returns new locations for objects, and does not detect alarms, etc., by it self. Alarms and similar would be detected by inquiries to the Object Database.
This solution would however cause a larger traffic load on the system, as all new location messages would have to be sent to the Object Database. Currently it is possible to set up the Location Server to only return new locations for an object if it, for instance, is moved more than five meters. If less logic were used in the Location Server, all new location messages would be returned to the Object Database.
The gain is that one does not have to replicate a lot of information. Also, the Object Database is made for this type of location data, and it might be the best place to store the information.

12.3 Map System

Suggestions for improvements in the current Map System are:

- Better support for showing maps in several floors.
- The current operation of the Map System is such that objects are drawn on the map before the map image is returned. This works fine for actors logged in to the system with role 2 and 3. However, for an actor with role 4 monitoring some objects, it would be annoying if the map image has to be reloaded every time an object is moved slightly. A better solution would be to separate the object icons from the map, and rather merge them at the actor's terminal. In this way the icons can be moved across the screen, without reloading the actual map image.
- All objects are drawn on the map using the same icon. It would be better if each type of object had its own icon. For instance, the location of a bed should be shown on the map using a bed-icon.
- Icons shown on the map should be clickable. When clicked, some information on the object, such as the current room number, etc., should be presented to the actor.

12.4 Object Database

The Object Database is the component that currently is best suited to the needs of the SAPT. All major things that the system requires from this component are supported by it. It is also a mature component. It has been employed in many similar systems as the SAPT for several years.

12.5 Routines at the hospital

To get a successful implementation of the SAPT at St. Olav's Hospital, many of the staffs' routines need to be changed. The most important ones are listed below:

- Staff need to realize the advantages of computers and technology, and do not oppose to it.
- Staff need to be registered when they arrive at work, and when they leave. The current role for each staff also needs to be recorded. This has to be done so that alarm messages, for instance, can be sent to the correct actors in the system.
- When some equipment is being used, this will have to be registered. This is needed to be able to search for 'free beds', for instance. After the equipment is no longer in use, the status for the equipment has to be se to 'free'.

How all these registration procedures shall be realized is a task that has to be solved. It is a very important one, as the system loses some of its value if it is not up to date at all time. Hopefully, staff at the hospital will realize that this is a gain, rather than a drawback.

13 Conclusion

The goal of this thesis was to find the requirements applying to a System for Alarm, Positioning and Tracking (SAPT), and to design this system. A small demonstrator implementing important concepts should also be made.

These tasks have been performed successfully. A requirements specification has been made for the system, along with a complete functional design. The system has been designed based on a detailed study of available techniques for positioning, and the systems that would surround the SAPT.
The implemented demonstrator has been tested in a prototype environment, and operated as intended.

The work with this thesis has provided the author with a better understanding of GIS systems and the concept of micro positioning. The thesis has also provided experience in one approach to systems engineering, which is based on first finding the requirements applying to a system, then creating a functional design for it, then making an implementation design, and finally implementing the system.
Even though only a demonstrator has been developed and not a complete system, it has provided a lot of programming exercise, and increased the author's ability to search for new solutions.

The fact that the results of this thesis might later be used by Bravida Geomatikk AS for a real implementation of a similar system at St. Olav's Hospital was a great motivator during the work with the thesis.

The main difficulty experienced in this study was to understand the system; what it was supposed to do and what should be able to perform by means of it. To handle the interaction and communication between the different components in an efficient way was also a major obstacle. Since the technologies used for micro positioning are rather new and improving frequently, it has been cumbersome to gather relevant information on these.

The author strongly believes that the system depicted here has a future to be installed and used in large buildings such as hospitals, museums, warehouses, etc. The application areas are

numerous; it can be used for tracking objects, finding objects, getting alarms, providing location aware services, facility management, etc.
Even though this thesis has focused around a hospital, and one concrete implementation, large parts of what has been found here can be reused in other installations.

14 Bibliography

[1] PCWebopedia: online dictionary, www.webopedia.com. Last visited June 10, 2004.

[2] Maneesh Prasad (2002); *Location based services*. http://www.gisdevelopment.net/technology/lbs/techlbs003.htm. Last visited May 24, 2004.

[3] V. Zeimpekis, G. M. Giaglis, G. Lekakos (2003); *A taxonomy of indoor and outdoor positioning techniques for mobile location services*, http://www.eltrun.aueb.gr/papers/Sigecom_3_4_final.pdf. Last visited January 29, 2004.

[4] O. Sæterbø (2003); *Innomhus GIS*. Unpublished document. Bravida Geomatikk AS.

[5] Oral communication with persons who know the St. Olavs Hospital in Trondheim, Norway.

[6] Helsebygg Midt-Norge- Nye St. Olavs Hospital (2003); *Bilag C, Funksjonsbeskrivelser med ansatteperspektiv i IKT-anlegg*. Interconsult ASA.

[7] Helsebygg Midt-Norge- Nye St. Olavs Hospital (2003); *Forespørsel 720-5002 Infrastruktur for IKT: B5 Kommunikasjonsanlegg*. Interconsult ASA.

[8] Press release by Radianse Indoor Positioning (2003); *Indoor Positioning Systems reach "Tipping Point:" Reduced cost and complexity expand application potential for healthcare*. http://www.radianse.com/Tipping%20Point.htm. Lase visited March 4, 2004.

[9] Versus technology, Inc. (2003); *Healthcare applications*. http://versustech.com/Healthcare_Applications/healthcare_apps.aspx. Last visited March 4, 2004.

[10] The Metro Group Future Store Initiative, http://www.future-store.org/servlet/PB/-s/1jhwupbszgcxtzu0ex1qgr899j2saf/menu/1002900_l2. Last visited March 6, 2004.

[11] Ekahau (2003); *SmartLibrary*. http://www.ekahau.com/pdf/SmartLibrary_Brochure.pdf. Last visited March 7, 2004.

[12] Sonitor Technologies (2003); *Evalutation report from Rikshospitalet, Oslo*. http://www.sonitor.com/news/article.asp?id=10. Last visited March 7, 2004.

[13] Odd R. Valmot (2003); *Radioøyet ser deg*. http://www.tu.no/ikt/article.jhtml?articleID=24182. Last visited March 10, 2004.

[14] M. L. Gjerding (2004); *Studenter laget mobilguide*. http://www.vg.no/pub/vgart.hbs?artid=228973. Last visited March 10, 2004.

[15] Garmin Emap GPS receiver, http://www.hardwaremaniac.com/reviews/emap/emap01.htm. Last visited January 31, 2004.

[16] D. Mountain, J. Raper (2001); *Positioning techniques for location-based services: characteristics and limitations of proposed solutions*, http://www.soi.city.ac.uk/~kam/positioningTechniquesForLBS.pdf. Last visited January 31, 2004.

[17] GSM World, http://www.gsmworld.com/technology/index.shtml. Last visited March 16, 2004.

[18] C. Smith, D. Collins (2002); *3G Wireless Networks*. McGraw-Hill.

[19] D. Porcino (2002); *Positioning techniques in radiocommunications standards*, http://www.telecom.ntua.gr/mobilevenue02/presentations/mlw_4_porcino.pdf. Last visited March 16, 2004.

[20] White paper from Radionor Communications, http://www.radionor.no/Downloads/radionor_white_paper.pdf. Last visited February 1, 2004.

[21] S. Høseggen (2003); *Tilbud fra Telenor, 720-5002 Infrastruktur for IKT, Vedlegg B5.5: Alarmering og gjenfinning*. Bravida Geomatikk AS.

[22] Ekahau Inc. Home Page, www.ekahau.com. Last visited February 5, 2004.

[23] BluePrint Wi-Fi, issue 39 - 22 (May 2003); *Tracking with Wi-Fi tags*.

[24] Sonitor Technologies, Home Page, www.sonitor.com. Last visited May 12, 2004.

[25] White paper from Intermec; *RFID technology in retail*, http://www.aimglobal.org/technologies/rfid/resources/RFIDinRetailWPAIM.pdf. Last visited February 12, 2004.

[26] Association for automatic Identification and Mobility; *What is Radio Frequency identificaton (RFID)?*, http://www.aimglobal.org/technologies/rfid/what_is_rfid.asp. Last visited February 12, 2004.

[27] Entry in PCWebopedia, http://www.webopedia.com/TERM/s/servlet.html. Last visited March 30, 2004.

[28] L.Tyrihjell, E. Wahl (2003); *GeoWEB web tjenester.* Unpublished document. Bravida Geomatikk AS.

[29] Oral communication with persons at Bravida Geomatikk AS, Trondheim, Norway.

[30] Oral communication with persons at Telenor FoU, Trondheim, Norway.

[31] Helsebygg Midt-Norge- Nye St. Olavs Hospital (2003); *Forespørsel 720-5002 Infrastruktur for IKT: B12 Katalogtjeneste.* Interconsult ASA.

[32] R. Bræk, Ø. Haugen (1993); *Engineering real time systems*. Prentice Hall Europe.

[33] HiST, Avdeling for informatikk og e-læring, *Krav til programvare*, http://www.aitel.hist.no/fag/_pum/omkrav/omkrav.doc. Last visited April 15, 2004.

[34] Sinan Si Alhir (2003); *Learning UML*. O'Reilly.

[35] IEEE Std 830-1993 (1994); *IEEE Recommended Practice for Software Requirements Specifications*, http://ieeexplore.ieee.org/iel1/3114/8906/00392555.pdf?isNumber=8906&prod=STD&arnumber=392555&arSt=i&ared=&arAuthor=. Last visited April 1, 2004.

[36] Tutorial on Microsoft PowerPoint, http://www.bcschools.net/staff/PowerPointHelp.htm. Last visited March 9, 2004.

[37] SDL Forum Society (2000); *What is SDL?*, http://www.sdl-forum.org/SDL/. Last visited April 23, 2004.

[38] C. Carlsson (2003); *TDA 142 I1 Programmering, Forelesning 6, Top-Down Design,* http://www.cs.chalmers.se/Cs/Grundutb/Kurser/i1pt/Forel/Forel6_OH.pdf. Last visited March 4, 2004.

[39] Tutorial on MSC, http://www.item.ntnu.no/fag/ttm4115/MSC_HTML-version/index.php. Last visited March 5, 2004.

[40] G. Fairhurst; *Asynchronous Communication*, http://www.erg.abdn.ac.uk/users/gorry/eg2069/async.html. Last visited March 15, 2004.

[41] C. Mellon, Software Engineering Institute; *Remote Procedure Call*, http://www.sei.cmu.edu/str/descriptions/rpc.html. Last visited March 30, 2004.

[42] Entry in PCWebopedia, http://www.pcwebopaedia.com/TERM/J/JSP.html. Last visited March 9, 2004.

[43] Java Technology home page, http://java.sun.com/. Last visited April 15, 2004.

[44] Eclipse project home page, http://www.eclipse.org/. Last visited April 1, 2004.

[45] Ericsson home page, http://www.ericsson.com/. Last visited April 23, 2004.

[46] Ø. Haugen, B. Møller-Pedersen (2000); *JavaFrame: Framework for Java Enabled Modelling*, http://www.item.ntnu.no/fag/SIE5065/Implementation/ECSE2000JavaFrame.pdf. Last visited April 23, 2004.

[47] Java 2 Platform, Standard Edition, v 1.3.1; *API Specification*, http://java.sun.com/j2se/1.3/docs/api/. Last visited May 2, 2004.

[48] J2EE; *The JDBC technology*, http://java.sun.com/products/jdbc/. Last visited May 2, 2004.

[49] J2EE; *Java Message Service (JMS)*, http://java.sun.com/products/jms/. Last visited May 2, 2004.

[50] SourceForge; *Open JMS*, http://openjms.sourceforge.net/usersguide/jars.html. Last visited May 2, 2004.

[51] Compendium in subject SIF8035: Informasjonssystemer, part 1, article P12 (2002); *Use Cases: Requirements in context*. Spring 2002, IDI, NTNU.

Appendix A Text based filled use cases - notation

This appendix provides a short introduction to the concept of text based filled use cases. First, the term ‘use case’ is described, and then a brief introduction to the filled form will be given. This appendix is based on [51].

What is a use case?
A use case is mainly a typical interaction between a user and a system. It is a set of scenarios that are connected through a common user target. Some of the characteristics of use cases are:

- a use case intercepts a function visible to the user
- a use case can be big or small
- a use case obtain a discrete target for the user

Filled use case:
A typical schema for a use case is given in table A–1 below.

The term ‘filled use case’ refers to the iteration of the text based use case. The iteration is a property of the use case, and is placed in the second row in the table below. The iteration ‘filled’ simply means that the all the properties of the text based use case are filled; i.e.: something is written in all the rows in the table. As long as one is working on the use case, more and more properties by it will be described through iterations. A use case that is not fully described is thus not ‘filled’. When one is working on a requirements specification, it is the filled version of the use case that is interesting, and therefore only those that are cited in the requirements specification.
The goal for the filled iteration is to provide comprehensive set of use cases and business rules that describes an application or a system. Even if these requirements are unpolished, they contain many details. The filled iteration contains many characteristics to choose from to make an informative use case. In table A–1, the characteristics used in this study are found, along with a description.

Characteristics:	Description:
Use case name	The name of the use case. Usually the primary user's goal is described. It is on a verb-substantive form
Iteration	Tells which detailing level is reached. A 'filled' iteration means that the user case is finished. A use case that is not yet finished is of the iteration 'façade'
Summary	A short description of the interaction between the system and the user
Basic event path	Describes the user's interaction with the system and other parties to achieve the user's goals. The basic event path is described as numbered instructions that tell what the user shall do, and how the system reacts to this. The assumption here is that the use case is successfully executed; i.e. that there are no errors
Alternative paths	The assumption is here is still that the use case is successfully executed, but one can achieve the goal via an alternative path
Exceptional paths	Describes the paths that are chosen when an error occurs, and the use case fails
Trigger	A use case trigger is an impulse or event that initiates the use case
Pre condition	A pre condition is everything that a use case assumes has happened before it is initiated. 'Everything' means conditions that are relevant for the use case
Post condition	A post condition is everything that a use case assumes has happened after is has finished. 'Everything' means conditions that are relevant for the use case
Author	The name of the person that has written the use case
Date	The date that the use case last was altered

Table A–1: Characteristics of the filled use case, with description

Appendix B Graphical interpretation of use cases

This appendix will give a graphical interpretation of the requirements specified in chapter 6.4.1: 'Functional requirements relating to actors' and chapter 6.4.3: 'Functional requirements to the SAPT'.

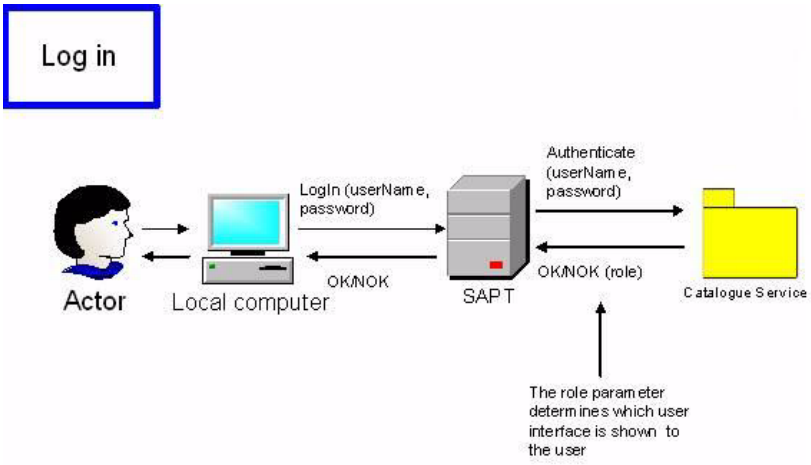


Figure B–1: Use case 'log in'

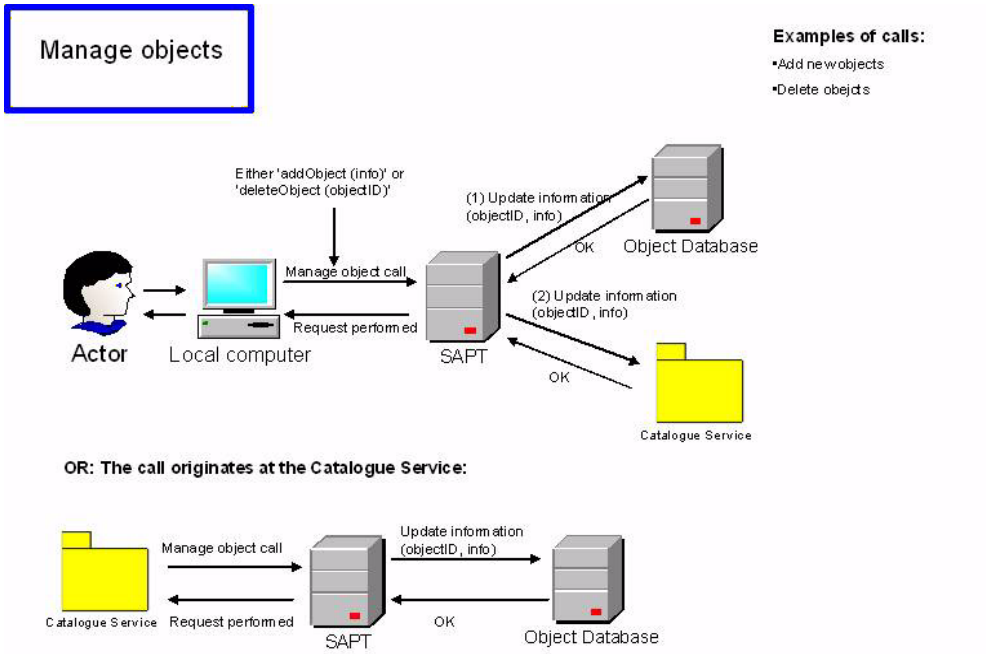


Figure B–2: Use case 'manage objects'

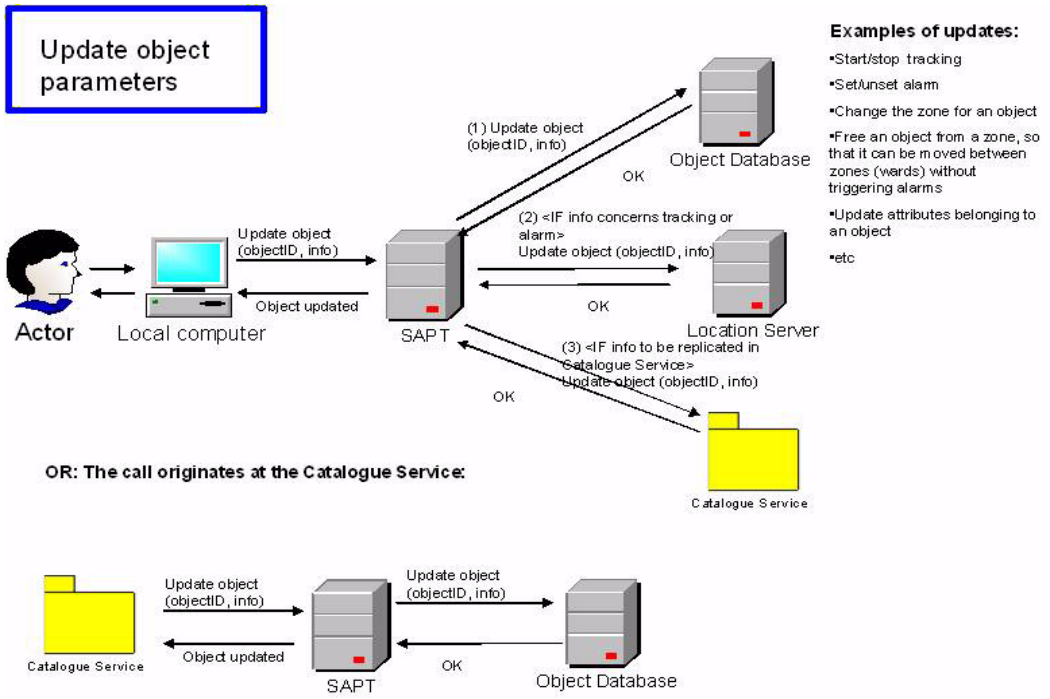


Figure B–3: Use case 'update object parameters'

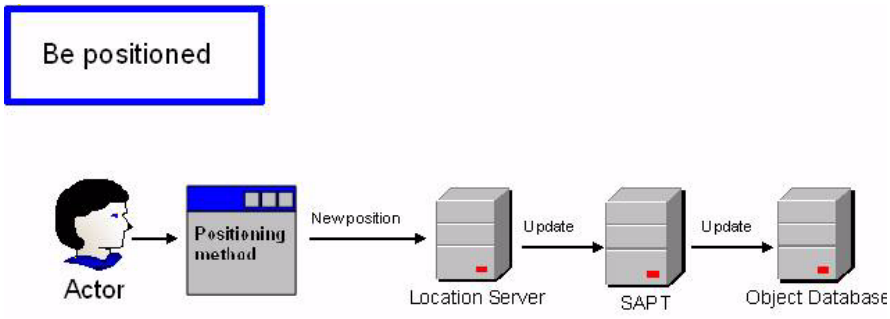


Figure B–4: Use case 'be positioned'

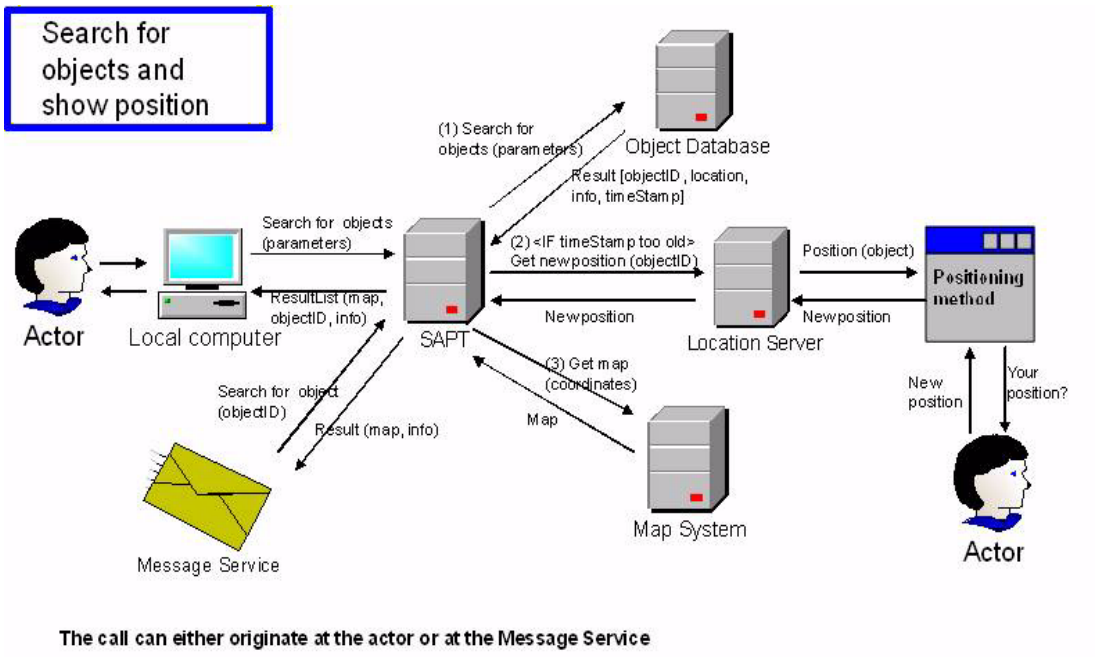


Figure B–5: Use case 'search for objects and show position'

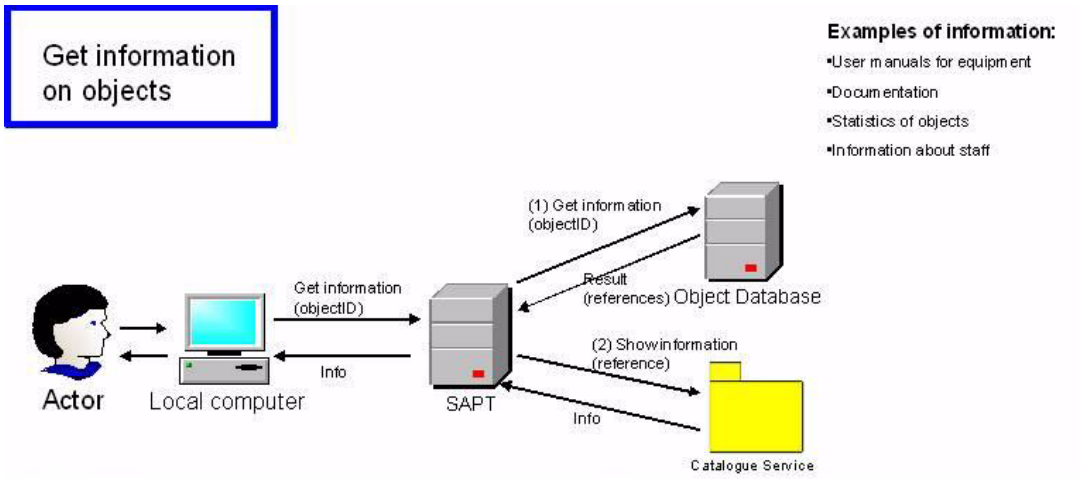


Figure B–6: Use case 'get information on objects'

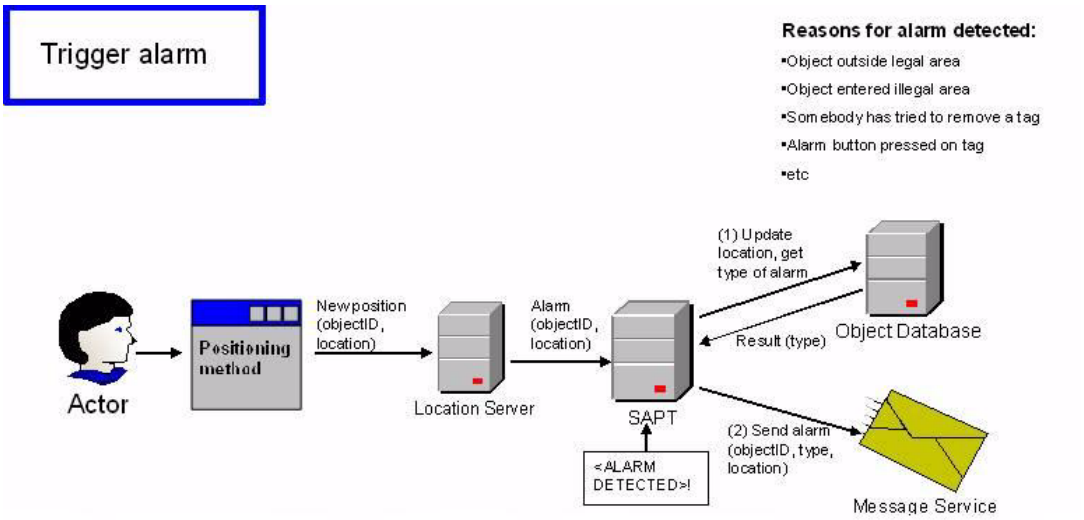


Figure B–7: Use case 'trigger alarm'

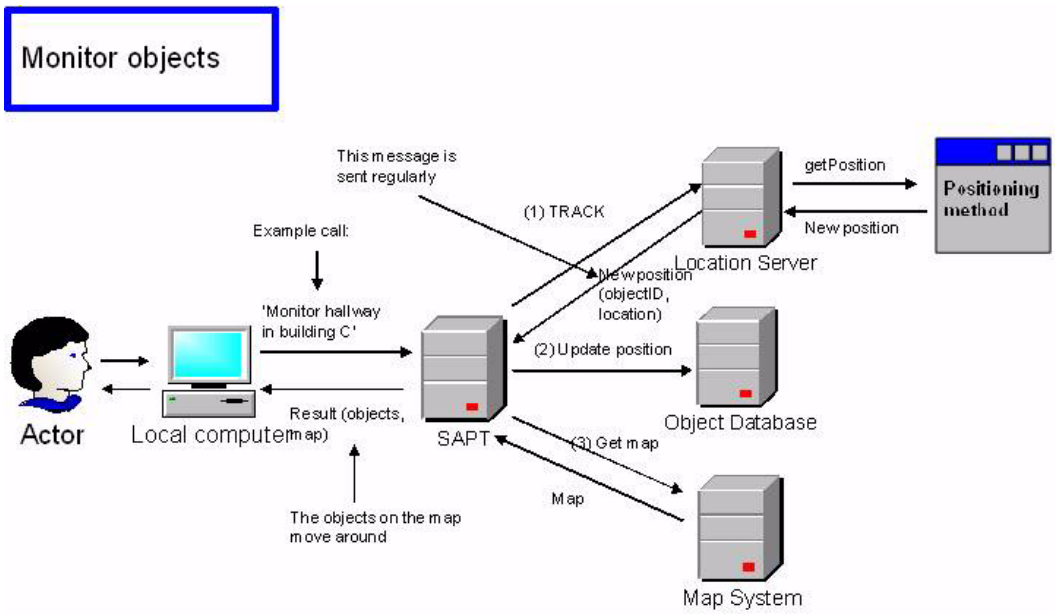


Figure B–8: Use case 'monitor objects'

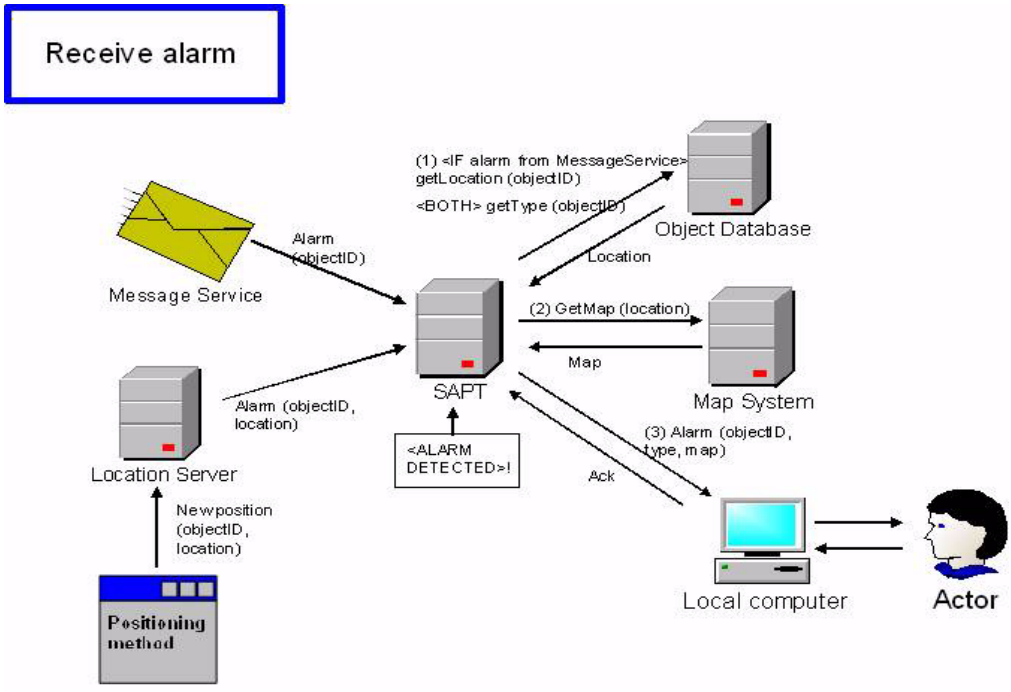


Figure B–9: Use case 'receive alarm'

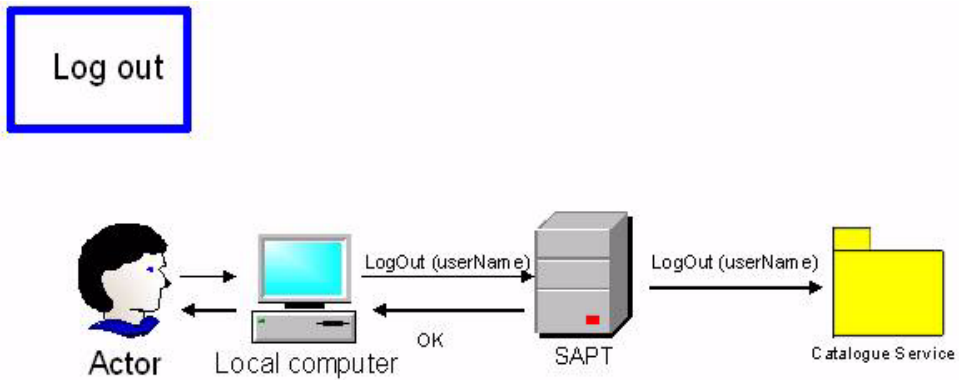


Figure B–10: Use case 'log out'

Appendix C JSP pages from prototype

This chapter cites the JSP pages that were made for the prototype developed in 2003. This is to give the reader a more detailed understanding of the system that is discussed in chapter 10: ’Evaluation of a prototype made in 2003’.
The chapter also shows how the following functional requirements are intended to be realized for actors logged in to the system with role 2:

- FA 1: Log in
- FA 5: Search for objects and show position of these
- FA 6: Get information on objects
- FA 10: Log out

The demonstration shows only one possible traversal of the program. There exist very many different traversals, depending on the parameters specified in the search procedure.

Figure C–1 shows the JSP page that allows actors to log in to the system.

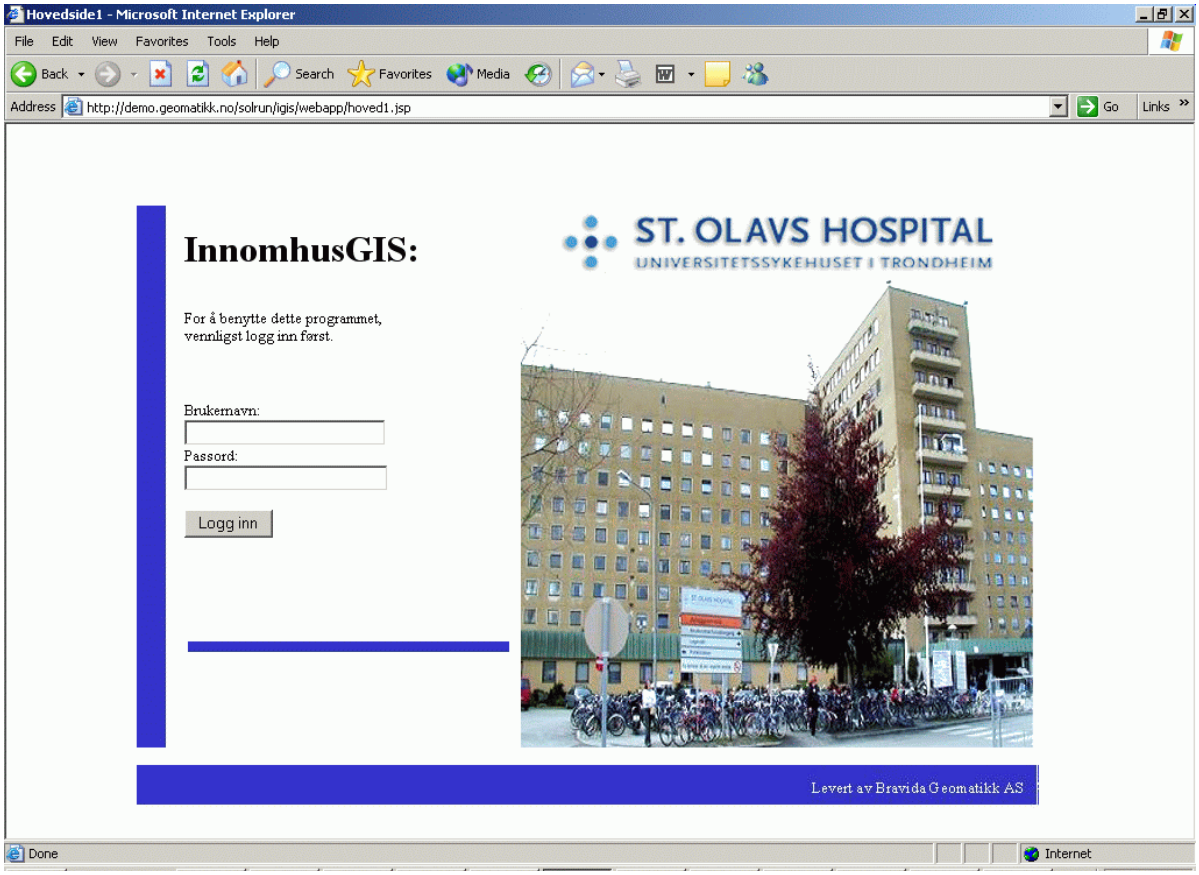


Figure C–1: JSP page 'log in page' from prototype

After logging in with username and password, the JSP page in figure C–2 is shown. This page makes it possible to search for objects matching certain parameters.

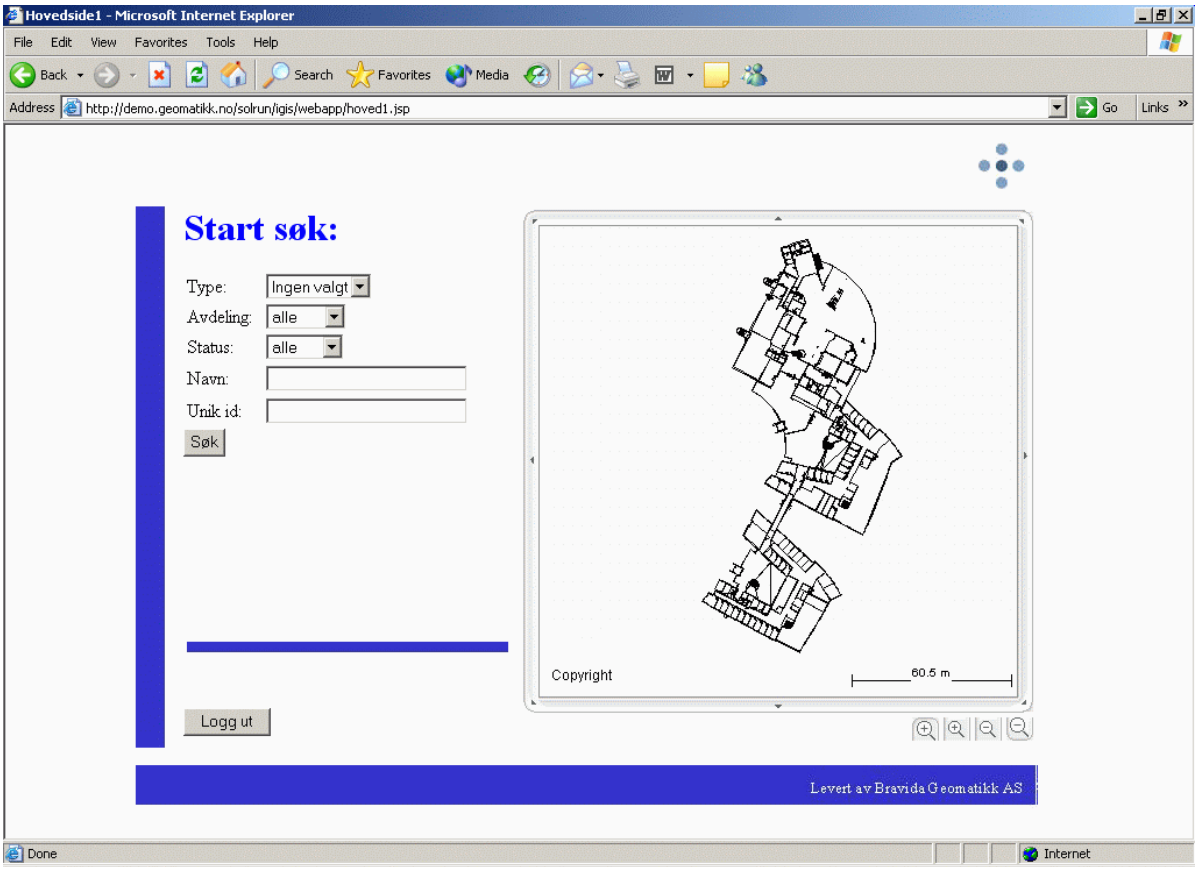


Figure C–2: JSP page 'search page' from prototype

The 'type' is set to 'seng', the 'avdeling' to 'alle' (default value), and 'status' to 'ledig'. The 'navn' and 'unik id' fields are left empty. The result when pressing the 'søk' button is shown in figure C–3.

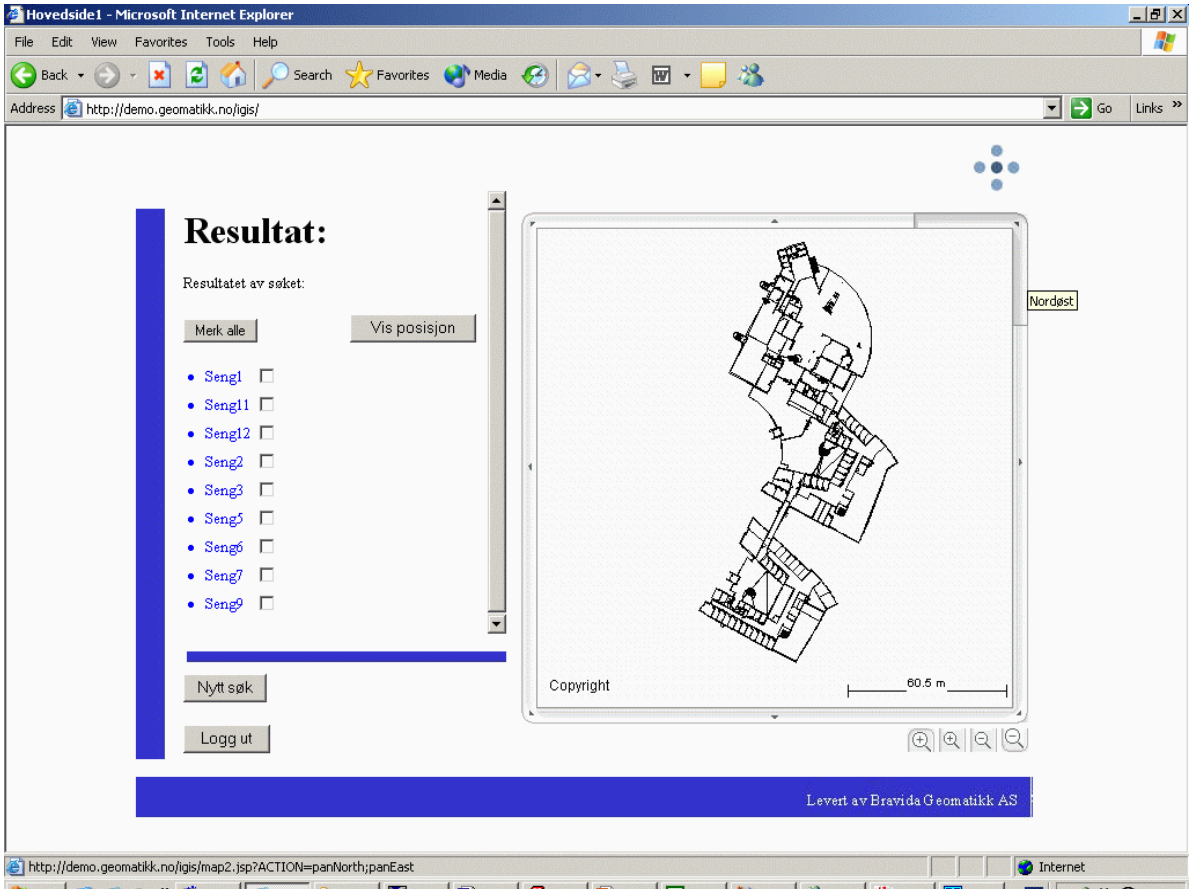


Figure C–3: JSP page 'result page' from prototype

The five objects at the top are selected from the list, and the 'vis posisjon' button is pressed. The page shown in figure C–4 is returned as a result.

.

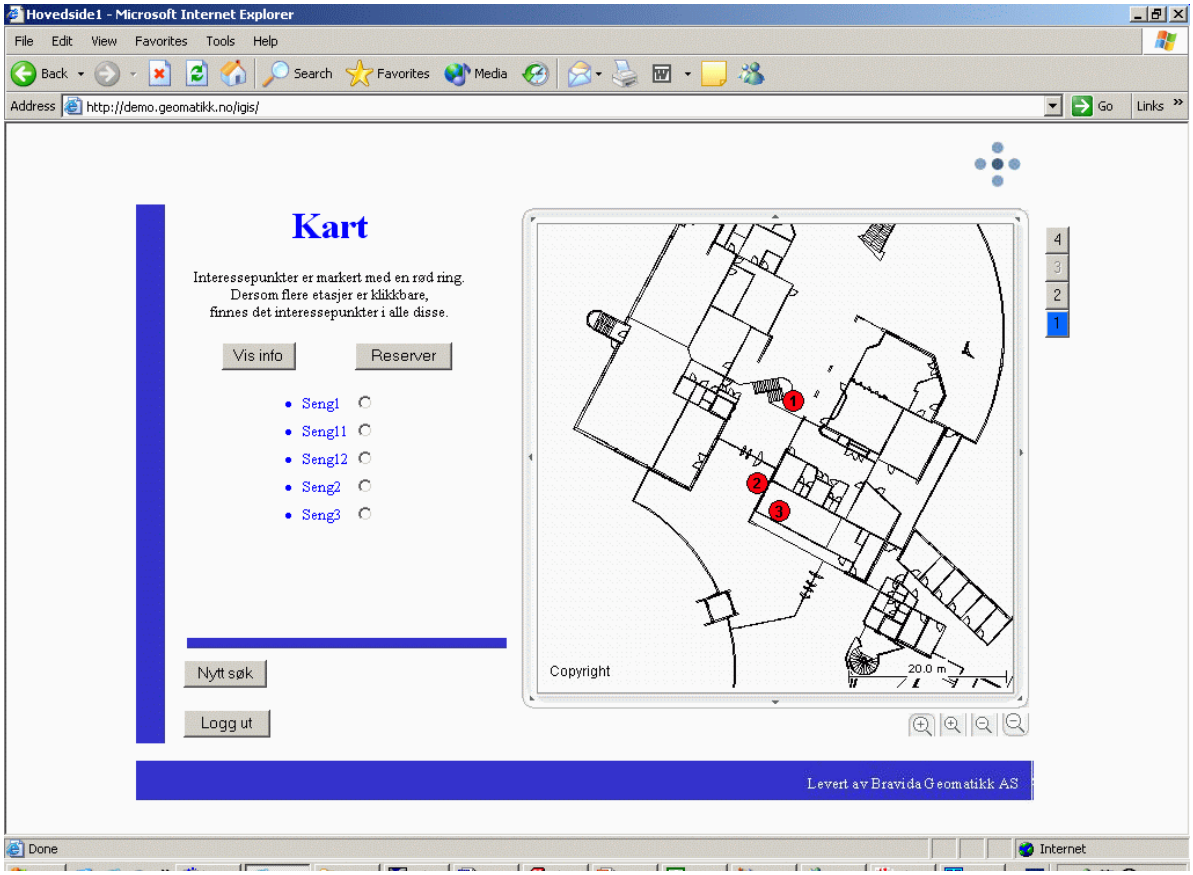


Figure C–4: JSP page 'map page' from prototype

The figure depicts that there are three beds on the first floor, shown with circles. That the shown beds are on the first floor can be seen on the button-row on the right hand side of the figure. The button-row shows that there are four floors on the building, and the first floor is marked with blue. I.e., it is the first floor that is currently being viewed.
It can also be seen that there are beds on two other floors, namely the 2nd and the 4th floor, since these buttons are clickable. There are no beds on the 3rd floor.
The objects shown on the map has a 'mouse-over' function that shows which bed a selected circle is representing if the mouse is moved over it.
On the left part of the screen, the objects selected on the previous page are shown in a list. Two buttons making it possible to either 'show info' on an object, or to 'reserve' an object is also shown. The reserve function is not implemented.

'Seng1' is then selected, and the 'show info' button is pressed. The result is shown in figure C–5.

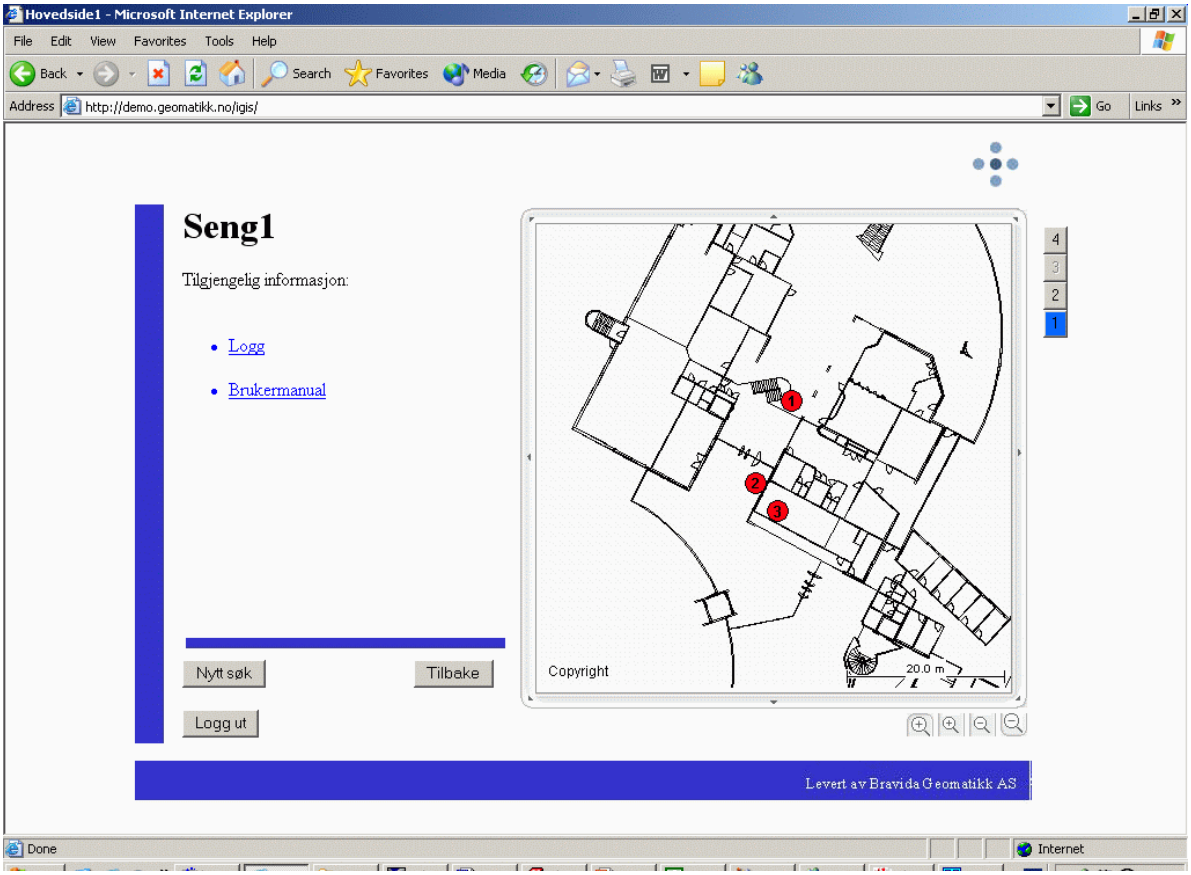


Figure C–5: JSP page 'info page' from prototype

As can be seen in the figure, there are two possible sources for information for this object. These are a log and a user manual. If one of the links on the page is pressed, currently only a 'dummy' page pops up, representing the information.

Appendix D Process graphs

This appendix cites the SDL process graphs for the components TerminalAgent, RouterBlock and LocationAgent.
The graphs are in accordance with that implemented for the SAPTdemo, and not with the designed 'ideal' system.

Figure D–1 shows the process graph for the implemented TerminalAgent process.

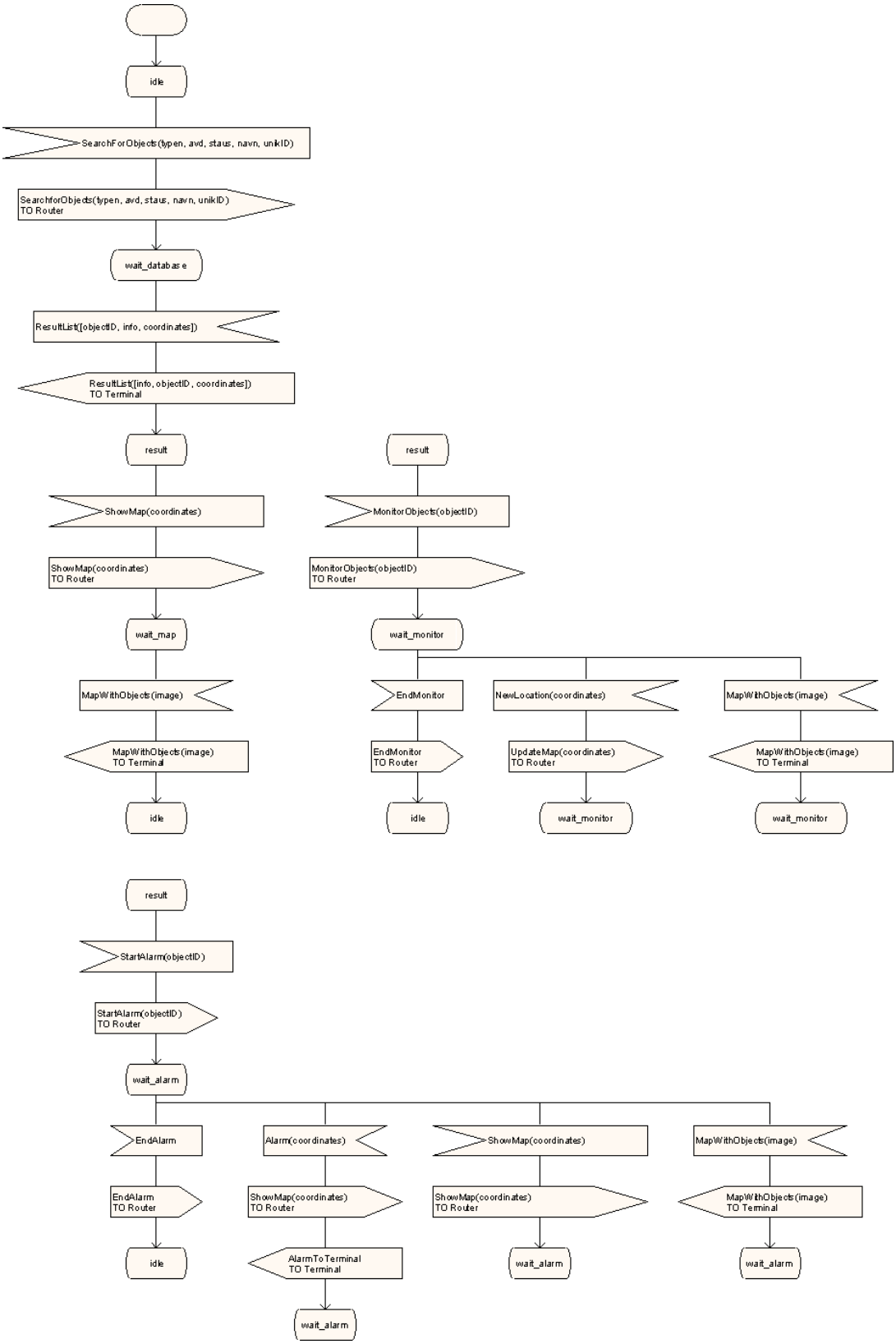


Figure D–1: Process graph for the implemented TerminalAgent

Figure D–2 shows the process graph for the implemented RouterBlock process.

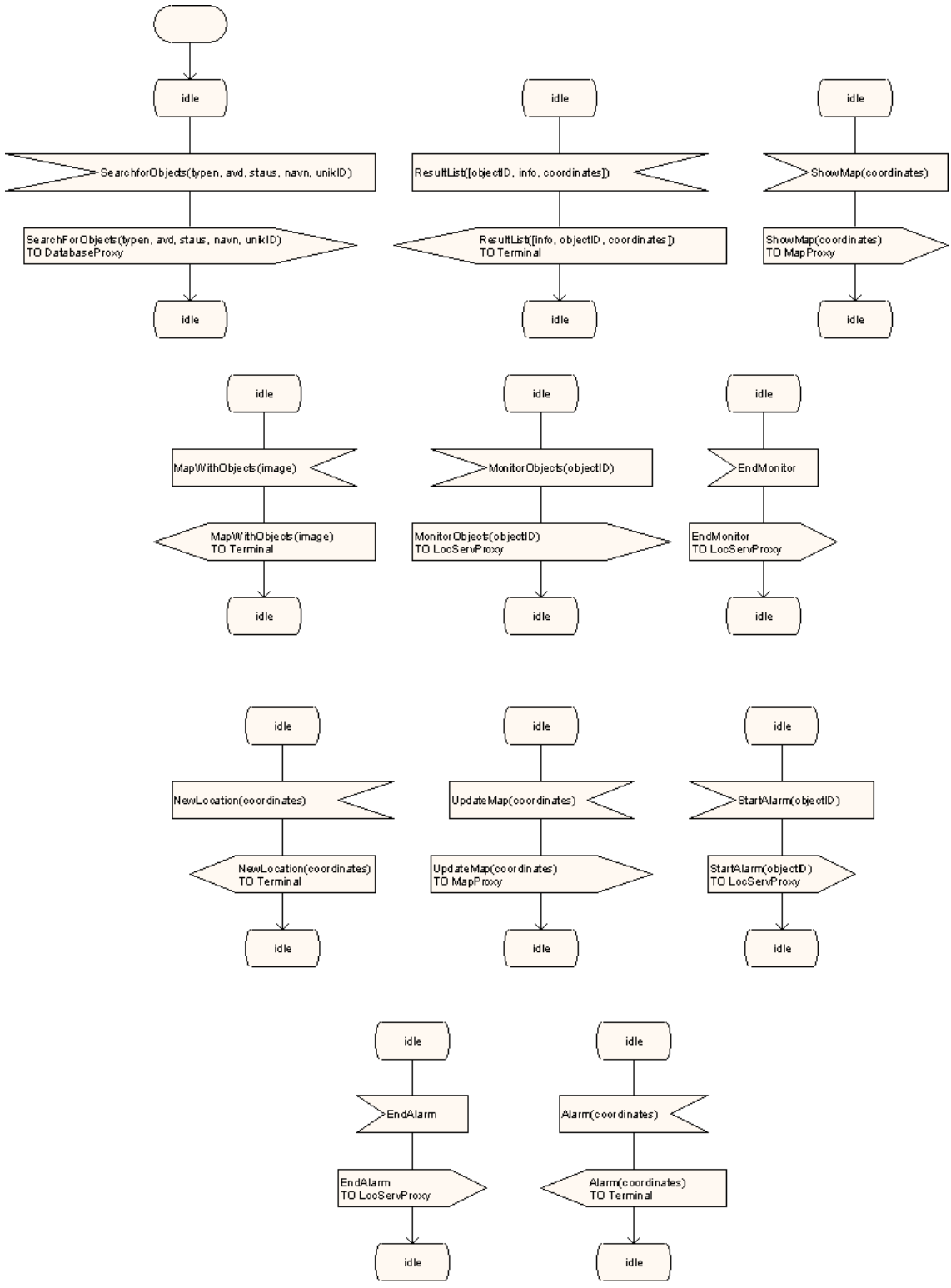


Figure D–2: Process graph for the implemented RouterBlock

Figure D–3 shows the process graph for the implemented LocationAgent process.

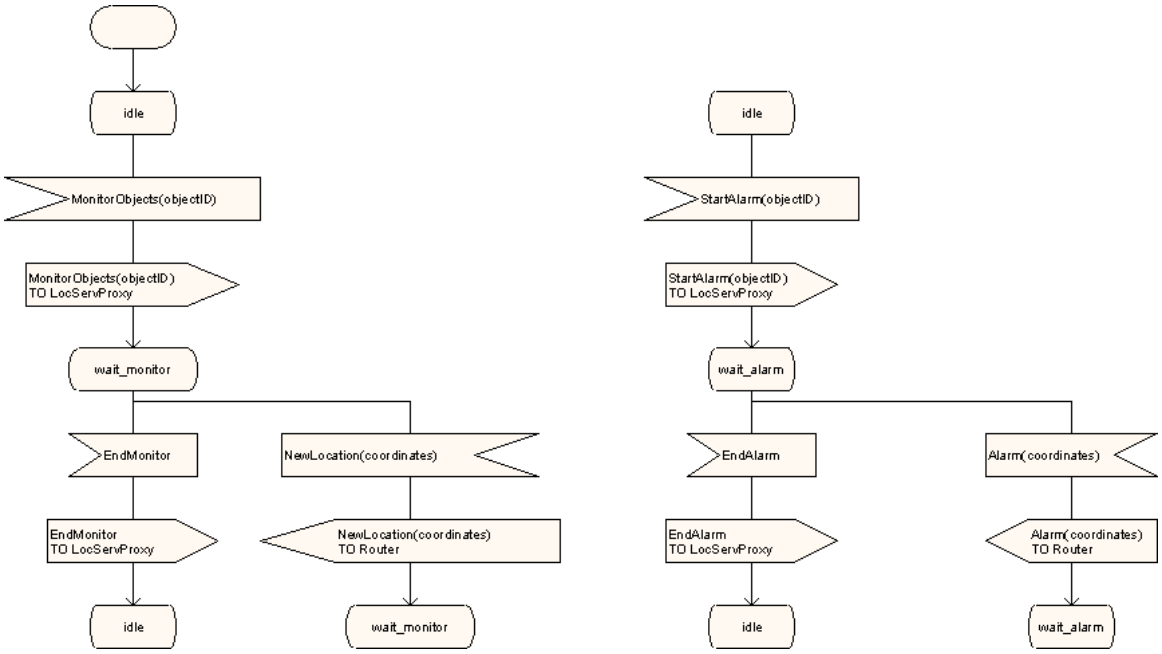


Figure D–3: Process graph for the implemented LocationAgent

Appendix E Installing and running

This appendix describes how to successfully install and run the demonstrator 'SAPTdemo' made in this thesis. The directory called SAPT can be found on the attached CD.

E.1 Installing

In the computer where the demonstrator is to be run:
- Install Java 2 Platform, Standard Edition (J2SE) version 1.3.1
- Copy the `SAPT` directory from the CD to the computer

The SAPT directory contains all the necessary code, along with the following needed jar files:
- `JFTrace.jar`
- `JavaFrame.jar`
- `jms_1.0.2a.jar`
- `openjms-client-0.7.4.jar`
- `oracle.sdoapi.jar`
- `exolabcore-0.3.4.jar`

Include these files in the CLASSPATH on the computer.

The following jar library also needs to be included in the CLASSPATH to run the demonstrator:
- `no.geomatikk.jar`

This library can be obtained from Bravida Geomatikk AS. It is not included on the CD because of confidentiality concerns.

Before running the demonstrator, make sure that you have connection to the following components:
- The Location Server at Telenor FoU
- The Object Database at Bravida Geomatikk AS

E.2 Running

Run the executable jar file `JFTrace.jar`:

- The file is found in `/SAPT/lib/`
- Double-click on the file to start it
- Choose File --> Open Input Socket
- Set input socket number to 54321
- Press OK

From a command window:

- Stand in directory `/SAPT/`
- Compile the code for all folders: `javac com.<name of folder>.*.java`
- Run the demonstrator with the command: `java com.controll.ExMain`

Appendix F System test

This appendix cites a test that has been performed on the SAPTdemo.

First, a plan that was set up prior to the test is presented. The next section gives a detailed description of the test, while the outprint that resulted from running the test is shown in the last section. The description is best understood if one at the same time studies the outprint.

F.1 Test plan

A plan was set up prior to the test. The plan is shown in table F–1.

Step	Description
1	Run the class ExMain.java
2	Search for all beds at the hospital
3	Show the location of one of the beds
4	Press the ‘nytt søk’ button
5	Search for all beds again
6	Monitor one of the beds
7	After about 5 minutes, press the ‘stopp monitor’ button
8	Search for all beds again
9	Set alarm for one of the beds
10	Wait until an alarm is received
11	Press the ‘OK’ button
12	Terminate the system

Table F–1: Test plan

F.2 Description

The test was performed on a Microsoft Windows machine at Bravida Geomatikk's facilities at Tyholt in Trondheim. Connection to the Object Database and the Location Server was confirmed prior to the test.
The test was carried out at May 21st, 2004, by Solrun Furuholt Pedersen. Steinar Høseggen, the supervisor of this thesis, monitored the test.

The SAPTdemo was started by running the class `ExMain`. This created all components, and connected the different mediators. The terminal GUI was displayed on the screen, looking like shown in figure F–1.

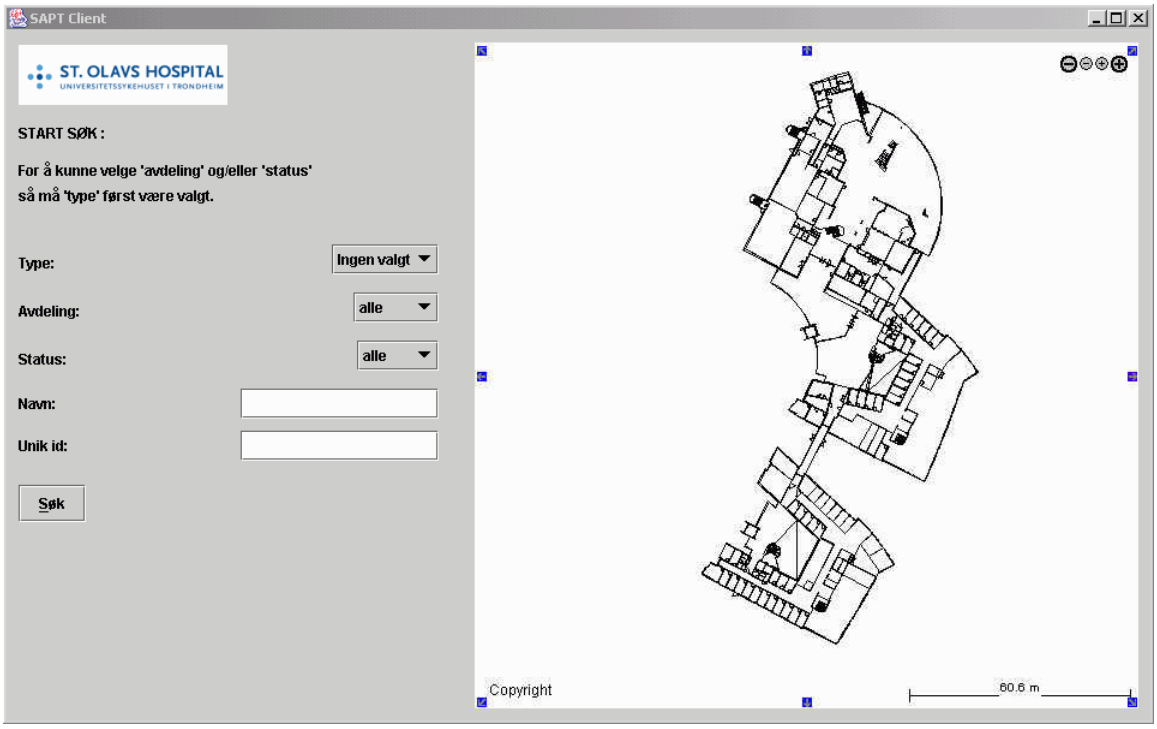


Figure F–1: 'Start search' view from SAPTdemo

A search for all beds at the hospital was then performed. The 'type' field was selected to be 'seng'. The other parameters were left untouched.

A message was then sent via the JavaFrame structure to the DatabaseProxy. The DatabaseProxy called the Object Database, which returned all objects matching the parameters. A list containing the name of the objects, their unique identification number and their location was sent to the terminal. The names of the objects were then presented in a list. This is indicated in figure F–2.

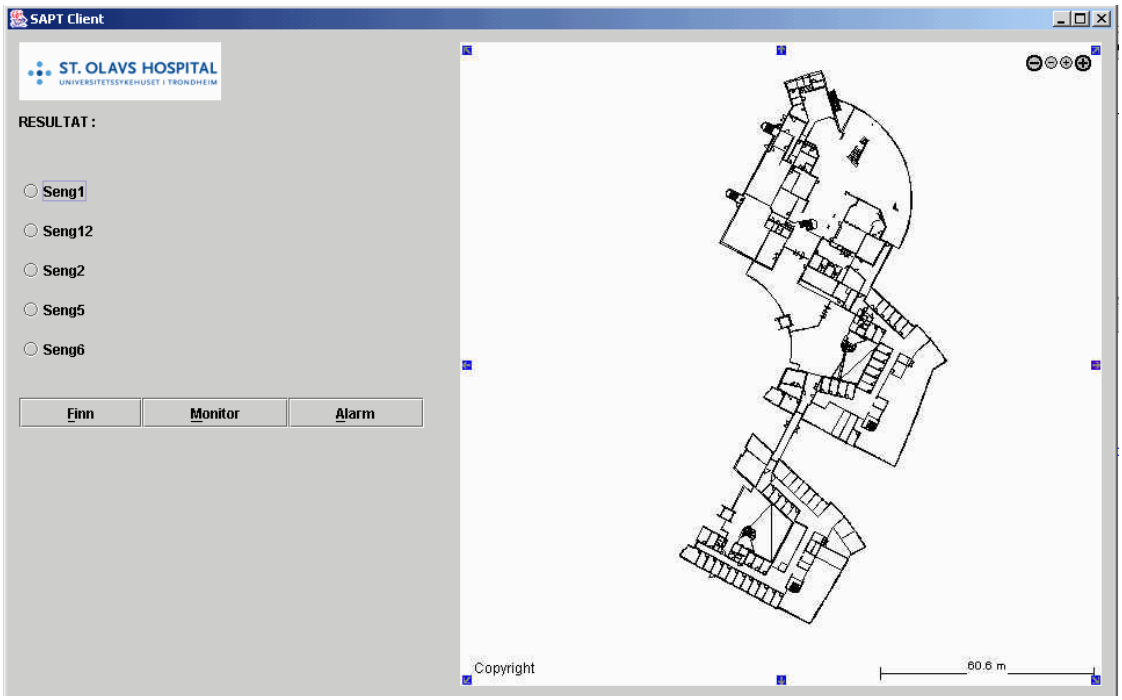


Figure F–2: 'Result' view from SAPTdemo

From the list, 'Seng5' was chosen, and the 'Finn' button was pressed. A message was sent via the JavaFrame structure to the MapProxy asking to view a map showing the selected object. The MapProxy returned a message containing the map, as shown in figure F–3.

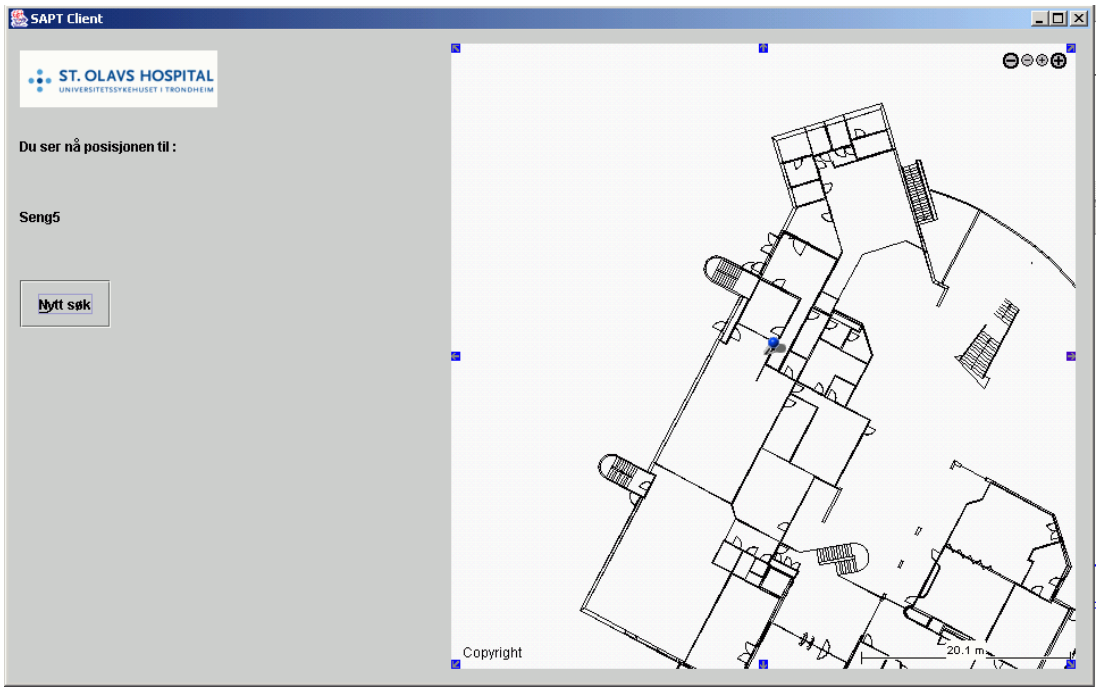


Figure F–3: 'Position' view from SAPTdemo

The location of ‘Seng 5’ is marked with a pin in the figure.

The button labelled ‘Nytt søk’ was pressed, and the view shown in figure F–1 was presented again. The search for all beds was repeated, and the view from figure F–2 was shown. This time ‘Seng12’ was selected, and the ‘Monitor’ button was pressed.
A message was then sent via the JavaFrame structure to the LocServProxy asking to start tracking the desired object. The LocServProxy forwarded this message to the Location Server. After this, messages from the Location Server with new locations for the object were returned to the TerminalAgent at regular intervals. The TerminalAgent sent requests to the MapProxy asking to get maps showing the new locations, and the new maps was returned to the terminal. A snapshot taken while monitoring is shown in figure F–4.

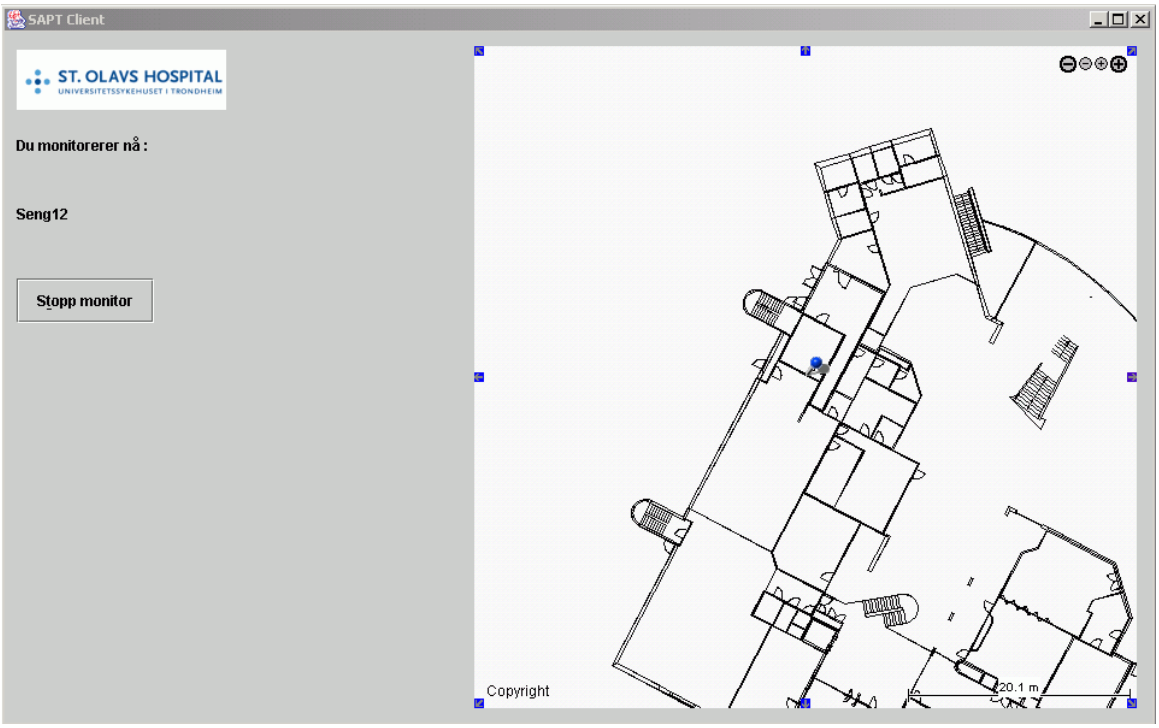


Figure F–4: ‘Monitoring’ view from SAPTdemo

After a while, the ‘Stopp monitor’ button was pressed, and a message was sent via the JavaFrame structure to the LocServProxy, asking to stop tracking the object. The terminal GUI returned to the view in figure F–1. The search for all beds was repeated, and the view from figure F–2 was shown again.

Now, ‘Seng6’ was selected, and the ‘Alarm’ button was pressed. A message was sent via the JavaFrame structure to the LocServProxy asking to start getting alarms for the object. A request was also sent to the MapProxy asking to view the current location of the object.
While waiting for alarm messages, the view of the terminal was like shown in figure F–5.

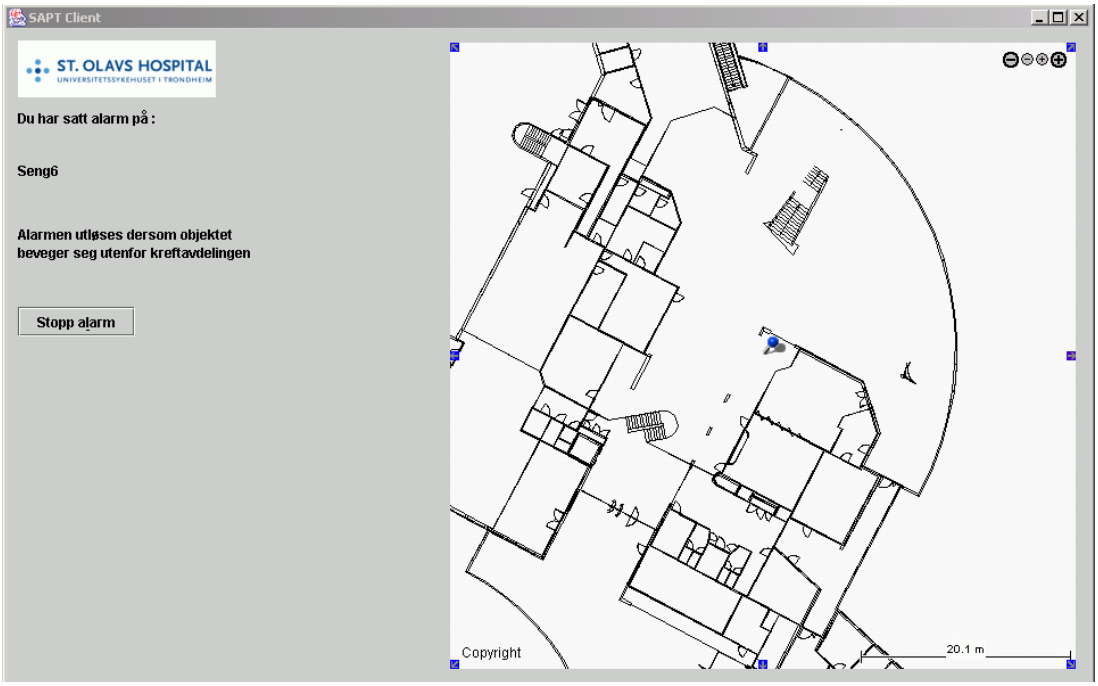


Figure F–5: 'Waiting alarm' view from SAPTdemo

After some waiting, an alarm was detected for the object by the Location Agent.
An alarm message was then sent to the terminal. A message was also sent to the MapProxy, asking it for an updated map with the location of the alarm. This resulting view is shown in figure F–6.

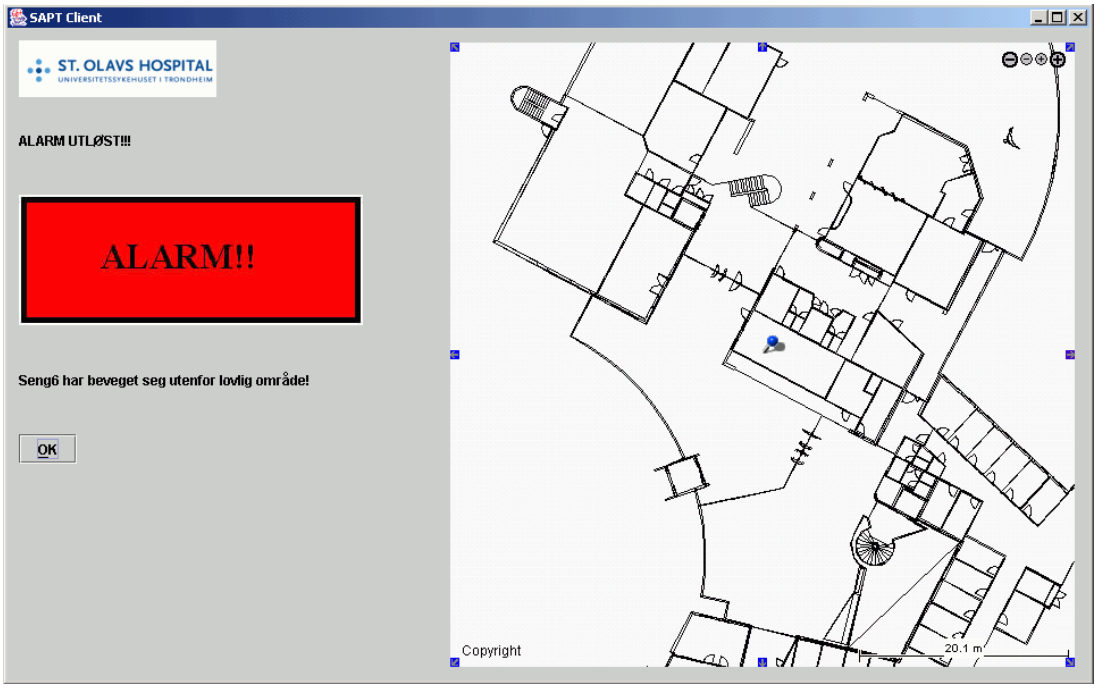


Figure F–6: 'Alarm' view from SAPTdemo

By then pressing the 'OK' button, the terminal view was reverted to that shown in figure F–1. A message was also sent via the JavaFrame structure to the LocServProxy, asking it to tell the Location Server to stop sending alarms for this object.

The test was then stopped.

A final comment on the outprint is that it can clearly be seen that the different messages are sent concurrently and asynchronously. The various active objects just produce messages, without waiting for the receiver to consume it.
What action that is taken upon receiving the different messages depends on the states of the active objects in the system. Some of the messages are just ignored (lost) if they are received in a state that does not handle them.

F.3 Outprint

```
Starting.........
Initalizing connection.....
Socket connection created.
Server connected to :  54321
**EdgeOutTerm created
**EdgeInTerm created
**EdgeInDb created
**EdgeOutDb created
**EdgeInMap created
**EdgeOutMap created
**EdgeInLocServ created
**EdgeOutLocServ created
**ComControll created
**Environment created
**DatabaseProxy: New DatabaseProxy created.
**MapProxy: New MapProxy created.
**LocServProxy: New LocServProxy created.
** JavaFrame starting **


Input parameters from the user:
type= seng, avd= alle, status= alle, navn= , unik id=

**EdgeInTerm: Sendt: SearchForObjectsM
**TermAgent(IdleTAS): Sendt: SearchForObjectsM
**Router(IdleR): Sendt: SearchForObjectsM
**EdgeOutDb: Sendt: SearchForObjectsM
**EdgeOutDb: parameters: sengallealle

DEBUG: no.geomatikk.database.Database - Getting database connection
'jdbc:oracle:thin:@10.254.10.142:1521:brapt', 'brapt', 'brapt'.
**DatabaseProxy: Sendt: ResultListM
**EdgeInDb: Sendt: ResultListM
**Router(IdleR): Sendt: ResultListM
**TermAgent(Wait_DatabaseTAS): Sendt: ResultListM
**EdgeOutTerm: Sendt: ResultListM
**Terminal: ResultList received:
-----------------------
Seng1,10057,10.4379445480347,63.4222959777832
-----------------------
Seng12,14015,10.4377775192261,63.422126770019496
-----------------------
Seng2,10058,10.4378808,63.4220678
-----------------------
Seng5,10059,10.4377012252808,63.4224319458008
-----------------------
```

```
Seng6,11693,10.4380693435669,63.4223175048828
-----------------------



Terminal: want to view the location of : Seng5
**EdgeInTerm: Sendt: ShowMapM
**TermAgent(ResultTAS): Sendt: ShowMapM
**Router(IdleR): Sendt: ShowMapM
**EdgeOutMap: Sendt: ShowMapM
**EdgeOutMap: parameters: xcoord=10.4377012252808 ycoord=63.4224319458008

**MapProxy: Received 10.4377012252808,63.4224319458008
**MapProxy: Map url is called: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.4377012252808&CY=63.4224319458008&ACTION=point&IW=550&IH=550&SC=500&MAP=864&
MAPUSER=Areal&MAPPASSWORD=Areal01

**MapProxy: Got a map. Loaded 22370 bytes from URL: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.4377012252808&CY=63.4224319458008&ACTION=point&IW=550&IH=550&SC=500&MAP=864&
MAPUSER=Areal&MAPPASSWORD=Areal01
**MapProxy: Sendt: MapWithObjectsM
**EdgeInMap: Sendt: MapWithObjectsM
**Router(IdleR): Sendt: MapWithObjectsM
**TermAgent(Wait_MapTAS): Sendt: MapWithObjectsM
**EdgeOutTerm: Sendt: MapWithObjectsM
**Terminal: Map received..



vi går tilbake...
Input parameters from the user:
type= seng, avd= alle, status= alle, navn= , unik id=

**EdgeInTerm: Sendt: SearchForObjectsM
**TermAgent(IdleTAS): Sendt: SearchForObjectsM
**Router(IdleR): Sendt: SearchForObjectsM
**EdgeOutDb: Sendt: SearchForObjectsM
**EdgeOutDb: parameters: sengallealle

DEBUG: no.geomatikk.database.Database - Getting database connection
'jdbc:oracle:thin:@10.254.10.142:1521:brapt', 'brapt', 'brapt'.
**DatabaseProxy: Sendt: ResultListM
**EdgeInDb: Sendt: ResultListM
**Router(IdleR): Sendt: ResultListM
**TermAgent(Wait_DatabaseTAS): Sendt: ResultListM
**EdgeOutTerm: Sendt: ResultListM
**Terminal: ResultList received:
-----------------------
Seng1,10057,10.4379445480347,63.4222959777832
-----------------------
Seng12,14015,10.4377775192261,63.422126770019496
-----------------------
Seng2,10058,10.4378808,63.4220678
-----------------------
Seng5,10059,10.4377012252808,63.4224319458008
-----------------------
Seng6,11693,10.4380693435669,63.4223175048828
-----------------------



Terminal: want to monitor  : Seng12. Unik ID er : 14015
**EdgeInTerm: Sendt: MonitorObjectsM
**TermAgent(ResultTAS): Sendt: MonitorObjectsM
**Router(IdleR): Sendt: MonitorObjectsM
**LocationAgent(IdleTAS): Sendt: MonitorObjectsM
**EdgeOutLocServ: Sendt: MonitorObjectsM
**EdgeOutLocServ: parameters: uId = 14015
```

```
**LocServProxy: Received: unikID : 14015. Action = monitor
**LocServProxy: TRACK request sendt
**LocServProxy: Waiting for result


**LocServProxy: Received position = coord: (63.42237, 10.437888), altitude: 3, type: test date:
Mon May 03 13:08:45 CEST 2004
x = 10.437888145446777
y = 63.42237091064453
z = 1.0
**LocServProxy: Sendt: NewLocationM. xcoord = 10.437888145446777 ycoord = 63.42237091064453

**EdgeInLocServ: Sendt: NewLocationM
**LocationAgent(Wait_MonitorTAS): Sendt: NewLocationM
**Router(IdleR): Sendt: NewLocationM
**TermAgent(Wait_MonitorTAS): Received: NewLocationM
**Router(IdleR): Sendt: ShowMapM
**EdgeOutMap: Sendt: ShowMapM
**EdgeOutMap: parameters: xcoord=10.437888145446777 ycoord=63.42237091064453

**MapProxy: Received 10.437888145446777,63.42237091064453
**MapProxy: Map url is called: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.437888145446777&CY=63.42237091064453&ACTION=point&IW=550&IH=550&SC=500&MAP=8
64&MAPUSER=Areal&MAPPASSWORD=Areal01


**LocServProxy: Received position = coord: (63.422543, 10.437737), altitude: 4, type: test date:
Mon May 03 13:08:55 CEST 2004
x = 10.437737464904785
y = 63.422542572021484
z = 1.0
**LocServProxy: Sendt: NewLocationM. xcoord = 10.437737464904785 ycoord = 63.422542572021484

**EdgeInLocServ: Sendt: NewLocationM

**MapProxy: Got a map. Loaded 25624 bytes from URL: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.437888145446777&CY=63.42237091064453&ACTION=point&IW=550&IH=550&SC=500&MAP=8
64&MAPUSER=Areal&MAPPASSWORD=Areal01
**MapProxy: Sendt: MapWithObjectsM
**EdgeInMap: Sendt: MapWithObjectsM
**Router(IdleR): Sendt: MapWithObjectsM
**LocationAgent(Wait_MonitorTAS): Sendt: NewLocationM
**TermAgent(Wait_MonitorTAS): Sendt: MapWithObjectsM
**EdgeOutTerm: Sendt: MapWithObjectsM
**Terminal: Map received..



**Router(IdleR): Sendt: NewLocationM
**TermAgent(Wait_MonitorTAS): Received: NewLocationM
**Router(IdleR): Sendt: ShowMapM
**EdgeOutMap: Sendt: ShowMapM
**EdgeOutMap: parameters: xcoord=10.437737464904785 ycoord=63.422542572021484

**MapProxy: Received 10.437737464904785,63.422542572021484
**MapProxy: Map url is called: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.437737464904785&CY=63.422542572021484&ACTION=point&IW=550&IH=550&SC=500&MAP=
864&MAPUSER=Areal&MAPPASSWORD=Areal01


**LocServProxy: Received position = coord: (63.42249, 10.437758), altitude: 3, type: test date:
Mon May 03 13:09:05 CEST 2004
x = 10.437758445739746
y = 63.422489166259766
z = 1.0
**LocServProxy: Sendt: NewLocationM. xcoord = 10.437758445739746 ycoord = 63.422489166259766
```

```
**EdgeInLocServ: Sendt: NewLocationM

**MapProxy: Got a map. Loaded 18938 bytes from URL: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.437737464904785&CY=63.422542572021484&ACTION=point&IW=550&IH=550&SC=500&MAP=
864&MAPUSER=Areal&MAPPASSWORD=Areal01
**MapProxy: Sendt: MapWithObjectsM
**EdgeInMap: Sendt: MapWithObjectsM
**Router(IdleR): Sendt: MapWithObjectsM
**LocationAgent(Wait_MonitorTAS): Sendt: NewLocationM
**TermAgent(Wait_MonitorTAS): Sendt: MapWithObjectsM
**EdgeOutTerm: Sendt: MapWithObjectsM
**Terminal: Map received..



**Router(IdleR): Sendt: NewLocationM
**TermAgent(Wait_MonitorTAS): Received: NewLocationM
**Router(IdleR): Sendt: ShowMapM
**EdgeOutMap: Sendt: ShowMapM
**EdgeOutMap: parameters: xcoord=10.437758445739746 ycoord=63.422489166259766

**MapProxy: Received 10.437758445739746,63.422489166259766
**MapProxy: Map url is called: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.437758445739746&CY=63.422489166259766&ACTION=point&IW=550&IH=550&SC=500&MAP=
864&MAPUSER=Areal&MAPPASSWORD=Areal01


**LocServProxy: Received position = coord: (63.42241, 10.437997), altitude: 4, type: test date:
Mon May 03 13:09:15 CEST 2004
x = 10.437996864318848
y = 63.42240905761719
z = 1.0
**LocServProxy: Sendt: NewLocationM. xcoord = 10.437996864318848 ycoord = 63.42240905761719

**EdgeInLocServ: Sendt: NewLocationM


**LocServProxy: Received position = coord: (63.422447, 10.4377), altitude: 1, type: test date:
Mon May 03 13:09:25 CEST 2004
x = 10.437700271606445
y = 63.422447204589844
z = 1.0
**LocServProxy: Sendt: NewLocationM. xcoord = 10.437700271606445 ycoord = 63.422447204589844

**EdgeInLocServ: Sendt: NewLocationM

**MapProxy: Got a map. Loaded 20975 bytes from URL: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.437758445739746&CY=63.422489166259766&ACTION=point&IW=550&IH=550&SC=500&MAP=
864&MAPUSER=Areal&MAPPASSWORD=Areal01
**MapProxy: Sendt: MapWithObjectsM
**EdgeInMap: Sendt: MapWithObjectsM
**Router(IdleR): Sendt: MapWithObjectsM
**LocationAgent(Wait_MonitorTAS): Sendt: NewLocationM
**LocationAgent(Wait_MonitorTAS): Sendt: NewLocationM
**TermAgent(Wait_MonitorTAS): Sendt: MapWithObjectsM
**EdgeOutTerm: Sendt: MapWithObjectsM
**Terminal: Map received..



**Router(IdleR): Sendt: NewLocationM
**Router(IdleR): Sendt: NewLocationM
**TermAgent(Wait_MonitorTAS): Received: NewLocationM
**TermAgent(Wait_MonitorTAS): Received: NewLocationM
**Router(IdleR): Sendt: ShowMapM
**EdgeOutMap: Sendt: ShowMapM
**EdgeOutMap: parameters: xcoord=10.437996864318848 ycoord=63.42240905761719
```

```
**MapProxy: Received 10.437996864318848,63.42240905761719
**MapProxy: Map url is called: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.437996864318848&CY=63.42240905761719&ACTION=point&IW=550&IH=550&SC=500&MAP=8
64&MAPUSER=Areal&MAPPASSWORD=Areal01


**LocServProxy: Received position = coord: (63.42242, 10.437979), altitude: 2, type: test date:
Mon May 03 13:09:35 CEST 2004
x = 10.437978744506836
y = 63.422420501708984
z = 1.0
**LocServProxy: Sendt: NewLocationM. xcoord = 10.437978744506836 ycoord = 63.422420501708984

**EdgeInLocServ: Sendt: NewLocationM


**LocServProxy: Received position = coord: (63.422123, 10.437925), altitude: 3, type: test date:
Mon May 03 13:09:45 CEST 2004
x = 10.437925338745117
y = 63.422122955322266
z = 1.0
**LocServProxy: Sendt: NewLocationM. xcoord = 10.437925338745117 ycoord = 63.422122955322266

**EdgeInLocServ: Sendt: NewLocationM

**MapProxy: Got a map. Loaded 24487 bytes from URL: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.437996864318848&CY=63.42240905761719&ACTION=point&IW=550&IH=550&SC=500&MAP=8
64&MAPUSER=Areal&MAPPASSWORD=Areal01
**MapProxy: Sendt: MapWithObjectsM
**EdgeInMap: Sendt: MapWithObjectsM
**Router(IdleR): Sendt: ShowMapM
**EdgeOutMap: Sendt: ShowMapM
**EdgeOutMap: parameters: xcoord=10.437700271606445 ycoord=63.422447204589844

**MapProxy: Received 10.437700271606445,63.422447204589844
**MapProxy: Map url is called: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.437700271606445&CY=63.422447204589844&ACTION=point&IW=550&IH=550&SC=500&MAP=
864&MAPUSER=Areal&MAPPASSWORD=Areal01


**LocServProxy: Received position = coord: (63.422188, 10.43799), altitude: 3, type: test date:
Mon May 03 13:09:55 CEST 2004
x = 10.437990188598633
y = 63.42218780517578
z = 1.0
**LocServProxy: Sendt: NewLocationM. xcoord = 10.437990188598633 ycoord = 63.42218780517578

**EdgeInLocServ: Sendt: NewLocationM

**MapProxy: Got a map. Loaded 21997 bytes from URL: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.437700271606445&CY=63.422447204589844&ACTION=point&IW=550&IH=550&SC=500&MAP=
864&MAPUSER=Areal&MAPPASSWORD=Areal01


**LocServProxy: Received position = coord: (63.422558, 10.43795), altitude: 4, type: test date:
Mon May 03 13:10:05 CEST 2004
x = 10.437950134277344
y = 63.42255783081055
z = 1.0
**LocServProxy: Sendt: NewLocationM. xcoord = 10.437950134277344 ycoord = 63.42255783081055

**EdgeInLocServ: Sendt: NewLocationM
**MapProxy: Sendt: MapWithObjectsM
**EdgeInMap: Sendt: MapWithObjectsM
**Router(IdleR): Sendt: MapWithObjectsM
**Router(IdleR): Sendt: MapWithObjectsM
**LocationAgent(Wait_MonitorTAS): Sendt: NewLocationM
**LocationAgent(Wait_MonitorTAS): Sendt: NewLocationM
```

```
**LocationAgent(Wait_MonitorTAS): Sendt: NewLocationM
**LocationAgent(Wait_MonitorTAS): Sendt: NewLocationM
**TermAgent(Wait_MonitorTAS): Sendt: MapWithObjectsM
**EdgeOutTerm: Sendt: MapWithObjectsM
**Terminal: Map received..



**TermAgent(Wait_MonitorTAS): Sendt: MapWithObjectsM
**EdgeOutTerm: Sendt: MapWithObjectsM
**Terminal: Map received..



**Router(IdleR): Sendt: NewLocationM
**Router(IdleR): Sendt: NewLocationM
**Router(IdleR): Sendt: NewLocationM
**Router(IdleR): Sendt: NewLocationM
**TermAgent(Wait_MonitorTAS): Received: NewLocationM
**TermAgent(Wait_MonitorTAS): Received: NewLocationM
**TermAgent(Wait_MonitorTAS): Received: NewLocationM
**TermAgent(Wait_MonitorTAS): Received: NewLocationM
**Router(IdleR): Sendt: ShowMapM
**EdgeOutMap: Sendt: ShowMapM
**EdgeOutMap: parameters: xcoord=10.437978744506836 ycoord=63.422420501708984

**MapProxy: Received 10.437978744506836,63.422420501708984
**MapProxy: Map url is called: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.437978744506836&CY=63.422420501708984&ACTION=point&IW=550&IH=550&SC=500&MAP=
864&MAPUSER=Areal&MAPPASSWORD=Areal01


**LocServProxy: Received position = coord: (63.42212, 10.43782), altitude: 1, type: test date:
Mon May 03 13:10:15 CEST 2004
x = 10.437820434570312
y = 63.422119140625
z = 1.0
**LocServProxy: Sendt: NewLocationM. xcoord = 10.437820434570312 ycoord = 63.422119140625

**EdgeInLocServ: Sendt: NewLocationM

Stopping monitor!
**EdgeInTerm: Sendt: EndMonitorM


**LocServProxy: Received position = coord: (63.422142, 10.43799), altitude: 2, type: test date:
Mon May 03 13:10:25 CEST 2004
x = 10.437990188598633
y = 63.422142028808594
z = 1.0
**LocServProxy: Sendt: NewLocationM. xcoord = 10.437990188598633 ycoord = 63.422142028808594

**EdgeInLocServ: Sendt: NewLocationM

**MapProxy: Got a map. Loaded 24116 bytes from URL: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.437978744506836&CY=63.422420501708984&ACTION=point&IW=550&IH=550&SC=500&MAP=
864&MAPUSER=Areal&MAPPASSWORD=Areal01
**MapProxy: Sendt: MapWithObjectsM
**EdgeInMap: Sendt: MapWithObjectsM
**Router(IdleR): Sendt: ShowMapM
**EdgeOutMap: Sendt: ShowMapM
**EdgeOutMap: parameters: xcoord=10.437925338745117 ycoord=63.422122955322266

**MapProxy: Received 10.437925338745117,63.422122955322266
**MapProxy: Map url is called: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.437925338745117&CY=63.422122955322266&ACTION=point&IW=550&IH=550&SC=500&MAP=
864&MAPUSER=Areal&MAPPASSWORD=Areal01
```

```
**LocServProxy: Received position = coord: (63.422516, 10.4379835), altitude: 1, type: test date:
Mon May 03 13:10:35 CEST 2004
x = 10.437983512878418
y = 63.422515869140625
z = 1.0
**LocServProxy: Sendt: NewLocationM. xcoord = 10.437983512878418 ycoord = 63.422515869140625

**EdgeInLocServ: Sendt: NewLocationM


**LocServProxy: Received position = coord: (63.422493, 10.437772), altitude: 3, type: test date:
Mon May 03 13:10:45 CEST 2004
x = 10.437771797180176
y = 63.42249298095703
z = 1.0
**LocServProxy: Sendt: NewLocationM. xcoord = 10.437771797180176 ycoord = 63.42249298095703

**EdgeInLocServ: Sendt: NewLocationM

**MapProxy: Got a map. Loaded 26664 bytes from URL: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.437925338745117&CY=63.422122955322266&ACTION=point&IW=550&IH=550&SC=500&MAP=
864&MAPUSER=Areal&MAPPASSWORD=Areal01
**MapProxy: Sendt: MapWithObjectsM
**EdgeInMap: Sendt: MapWithObjectsM
**Router(IdleR): Sendt: ShowMapM
**EdgeOutMap: Sendt: ShowMapM
**EdgeOutMap: parameters: xcoord=10.437990188598633 ycoord=63.42218780517578

**MapProxy: Received 10.437990188598633,63.42218780517578
**MapProxy: Map url is called: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.437990188598633&CY=63.42218780517578&ACTION=point&IW=550&IH=550&SC=500&MAP=8
64&MAPUSER=Areal&MAPPASSWORD=Areal01


**LocServProxy: Received position = coord: (63.42234, 10.438096), altitude: 3, type: test date:
Mon May 03 13:10:55 CEST 2004
x = 10.438096046447754
y = 63.422340393066406
z = 1.0
**LocServProxy: Sendt: NewLocationM. xcoord = 10.438096046447754 ycoord = 63.422340393066406

**EdgeInLocServ: Sendt: NewLocationM


**LocServProxy: Received position = coord: (63.422466, 10.437929), altitude: 2, type: test date:
Mon May 03 13:11:05 CEST 2004
x = 10.437929153442383
y = 63.42246627807617
z = 1.0
**LocServProxy: Sendt: NewLocationM. xcoord = 10.437929153442383 ycoord = 63.42246627807617

**EdgeInLocServ: Sendt: NewLocationM

**MapProxy: Got a map. Loaded 26527 bytes from URL: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.437990188598633&CY=63.42218780517578&ACTION=point&IW=550&IH=550&SC=500&MAP=8
64&MAPUSER=Areal&MAPPASSWORD=Areal01
**MapProxy: Sendt: MapWithObjectsM
**EdgeInMap: Sendt: MapWithObjectsM
**Router(IdleR): Sendt: ShowMapM
**EdgeOutMap: Sendt: ShowMapM
**EdgeOutMap: parameters: xcoord=10.437950134277344 ycoord=63.42255783081055

**MapProxy: Received 10.437950134277344,63.42255783081055
**MapProxy: Map url is called: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.437950134277344&CY=63.42255783081055&ACTION=point&IW=550&IH=550&SC=500&MAP=8
64&MAPUSER=Areal&MAPPASSWORD=Areal01
Input parameters from the user:
```

```
type= seng, avd= alle, status= alle, navn= , unik id=

**EdgeInTerm: Sendt: SearchForObjectsM


**LocServProxy: Received position = coord: (63.4224, 10.437829), altitude: 2, type: test date:
Mon May 03 13:11:16 CEST 2004
x = 10.43782901763916
y = 63.422401428222656
z = 1.0
**LocServProxy: Sendt: NewLocationM. xcoord = 10.43782901763916 ycoord = 63.422401428222656

**EdgeInLocServ: Sendt: NewLocationM


**LocServProxy: Received position = coord: (63.422348, 10.438056), altitude: 3, type: test date:
Mon May 03 13:11:26 CEST 2004
x = 10.438055992126465
y = 63.42234802246094
z = 1.0
**LocServProxy: Sendt: NewLocationM. xcoord = 10.438055992126465 ycoord = 63.42234802246094

**EdgeInLocServ: Sendt: NewLocationM

**MapProxy: Got a map. Loaded 18982 bytes from URL: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.437950134277344&CY=63.42255783081055&ACTION=point&IW=550&IH=550&SC=500&MAP=8
64&MAPUSER=Areal&MAPPASSWORD=Areal01
**MapProxy: Sendt: MapWithObjectsM
**EdgeInMap: Sendt: MapWithObjectsM
**Router(IdleR): Sendt: MapWithObjectsM
**Router(IdleR): Sendt: MapWithObjectsM
**Router(IdleR): Sendt: MapWithObjectsM
**Router(IdleR): Sendt: MapWithObjectsM
**LocationAgent(Wait_MonitorTAS): Sendt: NewLocationM
**LocationAgent(Wait_MonitorTAS): Sendt: NewLocationM
**LocationAgent(Wait_MonitorTAS): Sendt: NewLocationM
**LocationAgent(Wait_MonitorTAS): Sendt: NewLocationM
**LocationAgent(Wait_MonitorTAS): Sendt: NewLocationM
**LocationAgent(Wait_MonitorTAS): Sendt: NewLocationM
**LocationAgent(Wait_MonitorTAS): Sendt: NewLocationM
**LocationAgent(Wait_MonitorTAS): Sendt: NewLocationM
**TermAgent(Wait_MonitorTAS): Sendt: EndMonitorM
**TermAgent(IdleTAS): Sendt: SearchForObjectsM
**Router(IdleR): Sendt: NewLocationM
**Router(IdleR): Sendt: NewLocationM
**Router(IdleR): Sendt: NewLocationM
**Router(IdleR): Sendt: NewLocationM
**Router(IdleR): Sendt: NewLocationM
**Router(IdleR): Sendt: NewLocationM
**Router(IdleR): Sendt: NewLocationM
**Router(IdleR): Sendt: NewLocationM
**Router(IdleR): Sendt: EndMonitorM
**Router(IdleR): Sendt: SearchForObjectsM
**EdgeOutDb: Sendt: SearchForObjectsM
**EdgeOutDb: parameters: sengallealle

DEBUG: no.geomatikk.database.Database - Getting database connection
'jdbc:oracle:thin:@10.254.10.142:1521:brapt', 'brapt', 'brapt'.
**DatabaseProxy: Sendt: ResultListM
**EdgeInDb: Sendt: ResultListM
**Router(IdleR): Sendt: ResultListM
**TermAgent(Wait_DatabaseTAS): Sendt: ResultListM
**EdgeOutTerm: Sendt: ResultListM
**Terminal: ResultList received:
-----------------------
Seng1,10057,10.4379445480347,63.4222959777832
-----------------------
Seng12,14015,10.4377775192261,63.422126770019496
```

```
----------------------
Seng2,10058,10.4378808,63.4220678
----------------------
Seng5,10059,10.4377012252808,63.4224319458008
----------------------
Seng6,11693,10.4380693435669,63.4223175048828
----------------------



**LocationAgent(Wait_MonitorTAS): Sendt: EndMonitorM
**EdgeOutLocServ: Sendt: EndMonitorM
**LocServProxy: Received: EndMonitorM. Sending DELETE to Location Server..


Terminal: want to get alarm for : Seng6. Unik ID er : 11693
**EdgeInTerm: Sendt: StartAlarmM
**TermAgent(ResultTAS): Sendt: StartAlarmM
**Router(IdleR): Sendt: StartAlarmM
**LocationAgent(IdleTAS): Sendt: StartAlarmM
**EdgeOutLocServ: Sendt: StartAlarmM
**EdgeOutLocServ: parameters: uId = 11693
**EdgeInTerm: Sendt: ShowMapM

**LocServProxy: Received: unikID : 11693. Action = alarm
**LocServProxy: NOTIFY request sendt
**LocServProxy: Waiting for result
**TermAgent: Sendt: ShowMapM
**Router(IdleR): Sendt: ShowMapM
**EdgeOutMap: Sendt: ShowMapM
**EdgeOutMap: parameters: xcoord=10.4380693435669 ycoord=63.4223175048828

**MapProxy: Received 10.4380693435669,63.4223175048828
**MapProxy: Map url is called: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.4380693435669&CY=63.4223175048828&ACTION=point&IW=550&IH=550&SC=500&MAP=864&
MAPUSER=Areal&MAPPASSWORD=Areal01

**MapProxy: Got a map. Loaded 25279 bytes from URL: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.4380693435669&CY=63.4223175048828&ACTION=point&IW=550&IH=550&SC=500&MAP=864&
MAPUSER=Areal&MAPPASSWORD=Areal01
**MapProxy: Sendt: MapWithObjectsM
**EdgeInMap: Sendt: MapWithObjectsM
**Router(IdleR): Sendt: MapWithObjectsM
**TermAgent: Sendt: MapWithObjectsM
**EdgeOutTerm: Sendt: MapWithObjectsM
**Terminal: Map received..




**LocServProxy: Received position = coord: (63.422115, 10.437869), altitude: 3, type: test date:
Mon May 03 13:12:13 CEST 2004
x = 10.43786907196045
y = 63.422115325927734
z = 1.0
**LocServProxy: Sendt: xcoord = 10.43786907196045 ycoord = 63.422115325927734

**EdgeInLocServ: Sendt: AlarmM
**LocationAgent(Wait_AlarmTAS): Sendt: AlarmM
**Router(IdleR): Sendt: AlarmM
**TermAgent: Received: AlarmM
**EdgeOutTerm: Sendt: AlarmToTerminalM
**Terminal: New alarm message received!

**Router(IdleR): Sendt: ShowMapM
**EdgeOutMap: Sendt: ShowMapM
**EdgeOutMap: parameters: xcoord=10.43786907196045 ycoord=63.422115325927734
```

```
**MapProxy: Received 10.43786907196045,63.422115325927734
**MapProxy: Map url is called: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.43786907196045&CY=63.422115325927734&ACTION=point&IW=550&IH=550&SC=500&MAP=8
64&MAPUSER=Areal&MAPPASSWORD=Areal01

**MapProxy: Got a map. Loaded 26658 bytes from URL: http://demo.geomatikk.no/servlet/
MapProvider?CX=10.43786907196045&CY=63.422115325927734&ACTION=point&IW=550&IH=550&SC=500&MAP=8
64&MAPUSER=Areal&MAPPASSWORD=Areal01
**MapProxy: Sendt: MapWithObjectsM
**EdgeInMap: Sendt: MapWithObjectsM
**Router(IdleR): Sendt: MapWithObjectsM
**TermAgent: Sendt: MapWithObjectsM
**EdgeOutTerm: Sendt: MapWithObjectsM
**Terminal: Map received..



Stopping the alarm!
**EdgeInTerm: Sendt: EndAlarmM
**TermAgent: Sendt: EndAlarmM
**Router(IdleR): Sendt: EndAlarmM
**LocationAgent(Wait_AlarmTAS): Sendt: EndAlarmM
**EdgeOutLocServ: Sendt: EndAlarmM
**LocServProxy: Received: EndAlarmM. Sending DELETE to Location Server..
```

Appendix G Source code

In this appendix, the source code that has been written for this thesis is attached. Only the code files that the author of this thesis has written are given.
All code cited here, along with other code parts needed to run the demonstrator is available on the enclosed CD.

The different Java classes are separated by at line.

G.1 package com.controll

```
/*************************************************************
 * This is the main class that starts the program.
 * It contains the 'main' method.
 *
 * ExMain.java
 *
 * @author Solrun Pedersen
 *************************************************************/

package com.controll;

import se.ericsson.eto.norarc.javaframe.*;

import javax.swing.JFrame;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class ExMain {

        // The mediators declared for the main system
        Mediator termToEnv; //The mediator from TerminalAgent to Terminal
        Mediator toTermAgent;    //The mediator from Terminal to TerminalAgent
        Mediator routerToDb; //The mediator from Router to DatabaseProxy
        Mediator toRouterFromDb;   //The mediator from DatabaseProxy to Router
        Mediator routerToMap; //The mediator from Router to MapProxy
        Mediator toRouterFromMap; //The mediator from MapProxy to Router
        Mediator locToEnv;   //The mediator from LocationAgent to Location Server
        Mediator toLocAgent;  //The medator from Location Server to LocationAgent
```

```
        ComControl system;
        Terminal terminal;
        DatabaseProxy dbp;
        MapProxy mp;
        LocServProxy lsp;

        /**
        * Constructor --> creating the other components, such as the
        * Terminal GUI, the proxies, etc
        */
        public ExMain(String hostName,int portNr){

        //This object produces Trace information to file or stream for one scheduler.
        Trace trc = null;

        //This code is to create the state machine machinery, and to asociate
        //the Scheduler with a thread
            try{
                trc =new Trace(true,hostName,portNr);
                System.out.println("Server connected to :  "+ portNr);
                }
        catch(Exception e){
                        System.out.println("Feil ved Socket oppkobling");
        }

        //Associates this Scheduler with the given Trace generator object
        Scheduler scheduler = new Scheduler(trc);
        Thread t1 = new Thread(scheduler);

        // create edge mediators
        termToEnv = new EdgeOutTerm(this);
        toTermAgent = new EdgeInTerm(this);
        toRouterFromDb = new EdgeInDb(this);
        routerToDb = new EdgeOutDb(this);
        toRouterFromMap = new EdgeInMap(this);
        routerToMap = new EdgeOutMap(this);
        toLocAgent = new EdgeInLocServ(this);
        locToEnv = new EdgeOutLocServ(this);

        // create system
        system = new ComControl(scheduler, toTermAgent, termToEnv, toRouterFromDb, routerToDb,
toRouterFromMap, routerToMap, locToEnv, toLocAgent); // Lager hele systemet

        // create the environment
        terminal = new Terminal(toTermAgent);
                JFrame frame = new JFrame("SAPT Client");
                frame.addWindowListener(new WindowAdapter() {
                        public void windowClosing(WindowEvent e) {
                                terminal.disconnect();
                                System.exit(0);
                        }
                });
                frame.getContentPane().add("Center", terminal);
                frame.pack();
                frame.setVisible(true);

                //create the proxies
                dbp = new DatabaseProxy(toRouterFromDb);
                mp = new MapProxy(toRouterFromMap);
```

```
                lsp = new LocServProxy(toLocAgent);

        t1.start();
        System.out.println("** JavaFrame starting **");
        System.out.println("");
        System.out.println("");
        }

        /**
         * The main method.
         * Starts the whole system.
         */
        public static void main(String[] args) {
                System.out.println("Starting.........");
                ExMain exMain1 = new ExMain("127.0.0.1",54321);
        }
}
```

```
/*************************************************************
 * This is the main object in the system.
 * It will encounter the whole system structure:
 * TerminalAgent, RouterBlock and LocationAgent.
 *
 * ComControl.java
 *
 * @author Solrun Pedersen
 *************************************************************/

package com.controll;

import se.ericsson.eto.norarc.javaframe.*;

import com.locationserver.*;
import com.router.*;
import com.terminal.*;

public class ComControl extends Composite {

  public ComControl(Scheduler sched, Mediator toTermAgent, Mediator termToEnv, Mediator
toRouter, Mediator routerToDb, Mediator toRouterFromMap, Mediator routerToMap, Mediator
locToEnv, Mediator toLocAgent) {
    super();

    // create TerminalAgent with its mediators
    Mediator TaToEnv = new Mediator();
    Mediator TaToRout = new Mediator();
    Mediator TaToSelf = new Mediator();
    addActiveObject(new TerminalAgentSM(sched, TaToEnv, TaToRout, TaToSelf));

    // create RouterBlock with its mediators
    Mediator RoutToTa = new Mediator();
    Mediator RoutToLa = new Mediator();
    Mediator RoutToDb = new Mediator();
    Mediator RoutToMap = new Mediator();
    Mediator RoutToSelf = new Mediator();
    addActiveObject(new RouterBlockSM(sched, RoutToTa, RoutToLa, RoutToDb, RoutToMap,
RoutToSelf)); //Lager Router
```

```
        //      create LocationAgent with its mediators
        Mediator LaToEnv = new Mediator();
        Mediator LaToRout = new Mediator();
        Mediator LaToSelf = new Mediator();
        addActiveObject(new LocationAgentSM(sched, LaToEnv, LaToRout, LaToSelf));

    // Connect the outer mediators with the inner ones
        toTermAgent.addAddress(TaToSelf);
        TaToEnv.addAddress(termToEnv);
        toRouter.addAddress(RoutToSelf);
        RoutToDb.addAddress(routerToDb);
        toRouterFromMap.addAddress(RoutToSelf);
        RoutToMap.addAddress(routerToMap);
        toLocAgent.addAddress(LaToSelf);
        LaToEnv.addAddress(locToEnv);

    // Connect inner mediators with other inner mediators
        TaToRout.addAddress(RoutToSelf);
        RoutToTa.addAddress(TaToSelf);
        LaToRout.addAddress(RoutToSelf);
        RoutToLa.addAddress(LaToSelf);

    System.out.println("**ComControll created");
  }
}
```

```
/*************************************************************
 * This is the GUI that is presented to the actors
 * in the system. It also makes it possible for actors to
 * send messages to the rest of the system.
 *
 * Terminal.java
 *
 * @author Solrun Pedersen
 *************************************************************/

package com.controll;

import se.ericsson.eto.norarc.javaframe.*;
import com.messages.*;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.StringTokenizer;
import javax.swing.border.EmptyBorder;
import java.util.ArrayList;


public class Terminal extends JPanel {

        Mediator toTermAgent;

        static JFrame frame;
        private JEditorPane html;
        private JTextArea txtPosition;
```

```
Box outerOuterBox; //total box
Box outerbox;   //left side of the box
Box mapbox;     //right side of the box

ArrayList result;

//for the first view:
public String type = "Ingen valgt";
public String avd = "alle";
public String status = "alle";
public JTextField navnTextField;
public JTextField unikTextField;
public String navn;
public String unikId;

//for the second view:
public String selected = "";
public ButtonGroup group = new ButtonGroup();

//whether to change the left side or not:
public boolean left = true;

/**
 * Constructor --> creating the Terminal GUI
 */
public Terminal(Mediator toS) {
        toTermAgent = toS;

        // Create panels and layout
        this.setLayout(new BorderLayout());
        this.setBorder(new EmptyBorder(10, 10, 10, 10));
        outerOuterBox = Box.createHorizontalBox(); //Total box
        outerbox = Box.createVerticalBox(); //left side
        mapbox = Box.createVerticalBox(); //right side

        createLeftSide();
        outerOuterBox.add(outerbox);
        outerOuterBox.add(Box.createHorizontalStrut(30));

        // Add map
        createMap();
        outerOuterBox.add(mapbox);
        outerOuterBox.add(Box.createVerticalStrut(10));
        add(outerOuterBox, BorderLayout.CENTER);

        System.out.println("**Environment created");
}

/**
 * Method to create the left side of the GUI initially
 */
public void createLeftSide(){

        outerbox.add(createImg("stolav_logo.gif"));
        outerbox.add(Box.createVerticalStrut(10));

        outerbox.add(createText("START SØK :"));
```

```
                outerbox.add(Box.createVerticalStrut(10));
                outerbox.add(createText("For å kunne velge 'avdeling' og/eller 'status'"));
                outerbox.add(createText("så må 'type' først være valgt."));
                outerbox.add(Box.createVerticalStrut(30));

                //create search parameters
                outerbox.add(createType());
                outerbox.add(Box.createVerticalStrut(10));
                outerbox.add(createAvdeling());
                outerbox.add(Box.createVerticalStrut(10));
                outerbox.add(createStatus());
                outerbox.add(Box.createVerticalStrut(10));
                outerbox.add(createNavn());
                outerbox.add(Box.createVerticalStrut(10));
                outerbox.add(createUnik());
                outerbox.add(Box.createVerticalStrut(20));

                //create button
                outerbox.add(createSokButton());
                outerbox.add(Box.createVerticalStrut(150));
                outerbox.add(Box.createHorizontalStrut(350));
        }

        /**
         * Method to create the initial right side --> the map.
         * The first map is just a static image,
         * and is not gotten from the Map Server
         */
        private void createMap(){
                mapbox.removeAll();
                JPanel panel = new JPanel(new BorderLayout());
                ImageIcon icon = new ImageIcon("map.jpg");
                JLabel lbl = new JLabel(icon);
                panel.add(lbl, BorderLayout.WEST);
                mapbox.add(panel);
                mapbox.validate();
                mapbox.repaint();
        }

        /**
         * Method to update the left side of the GUI as
         * one receives new messages.
         * @param action Defines which action to take when updating the GUI
         */
        public void updateLeftSide(String action){
                outerbox.removeAll();
                outerbox.add(createImg("stolav_logo.gif"));
                outerbox.add(Box.createVerticalStrut(10));

                if(action.equals("sok")){
                        //Clear previous results:
                        result.clear();
                        selected = "";
                        left = true;

                        outerbox.add(createText("START SØK :"));
                        outerbox.add(Box.createVerticalStrut(10));
                        outerbox.add(createText("For å kunne velge 'avdeling' og/eller
'status'"));
                        outerbox.add(createText("så må 'type' først være valgt."));
```

```
                        outerbox.add(Box.createVerticalStrut(30));

                        //create search parameters
                        outerbox.add(createType());
                        outerbox.add(Box.createVerticalStrut(10));
                        outerbox.add(createAvdeling());
                        outerbox.add(Box.createVerticalStrut(10));
                        outerbox.add(createStatus());
                        outerbox.add(Box.createVerticalStrut(10));
                        outerbox.add(createNavn());
                        outerbox.add(Box.createVerticalStrut(10));
                        outerbox.add(createUnik());
                        outerbox.add(Box.createVerticalStrut(20));

                        //create button
                        outerbox.add(createSokButton());
                }
                else if(action.equals("resultat")){
                        boolean found = true;

                        outerbox.add(createText("RESULTAT :"));
                        outerbox.add(Box.createVerticalStrut(30));

                        //List result of search
                        for(int i=0; i<result.size(); i++){
                                String all = (String) result.get(i);
                                if(all.startsWith("Ingen")){
                                        outerbox.add(createText("Fant ingen objekter som matchet
ditt søk."));
                                        found = false;
                                }
                                else{
                                        int del = all.indexOf(",");
                                        String name = all.substring(0,del);
                                        outerbox.add(Box.createVerticalStrut(10));
                                        outerbox.add(createRadioButton(name));
                                }
                        }
                        if(found){
                                outerbox.add(Box.createVerticalStrut(30));
                                //create buttons
                                JButton btnFind = createFindButton();
                                JButton btnMonitor = createMonitorButton();
                                JButton btnAlarm = createAlarmButton();
                                Box buttonBox = Box.createHorizontalBox();
                                buttonBox.add(btnFind);
                                buttonBox.add(btnMonitor);
                                buttonBox.add(btnAlarm);
                                outerbox.add(buttonBox);
                                outerbox.add(Box.createVerticalStrut(100));
                        }
                        else{
                                outerbox.add(Box.createVerticalStrut(40));
                                outerbox.add(createBackButton());
                                outerbox.add(Box.createVerticalStrut(150));
                        }
                }
                else if(action.equals("finn")){
                        outerbox.add(createText("Du ser nå posisjonen til :"));
                        outerbox.add(Box.createVerticalStrut(30));
```

```
                        outerbox.add(createText(selected));
                        outerbox.add(Box.createVerticalStrut(40));
                        outerbox.add(createBackButton());
                        outerbox.add(Box.createVerticalStrut(150));
                }
                else if(action.equals("monitorer")){
                        outerbox.add(createText("Du monitorerer nå :"));
                        outerbox.add(Box.createVerticalStrut(30));
                        outerbox.add(createText(selected));
                        outerbox.add(Box.createVerticalStrut(40));
                        outerbox.add(createStopMonitorButton());
                        outerbox.add(Box.createVerticalStrut(170));
                }
                else if(action.equals("alarmer")){
                        outerbox.add(createText("Du har satt alarm på :"));
                        outerbox.add(Box.createVerticalStrut(30));
                        outerbox.add(createText(selected));
                        outerbox.add(Box.createVerticalStrut(40));
                        outerbox.add(createText("Alarmen utløses dersom objektet"));
                        outerbox.add(createText("beveger seg utenfor kreftavdelingen"));
                outerbox.add(Box.createVerticalStrut(40));
                        outerbox.add(createStopAlarmButton());
                        outerbox.add(Box.createVerticalStrut(150));
                }
                else if(action.equals("alarmNow")){
                        outerbox.add(Box.createVerticalStrut(20));
                        outerbox.add(createText("ALARM UTLØST!!!"));
                        outerbox.add(Box.createVerticalStrut(40));
                        outerbox.add(createImg("alarm.jpg"));
                        outerbox.add(Box.createVerticalStrut(40));
                        outerbox.add(createText(selected + " har beveget seg utenfor lovlig
område!"));
                        outerbox.add(Box.createVerticalStrut(40));
                        outerbox.add(createAckAlarmButton());
                outerbox.add(Box.createVerticalStrut(150));
                }

                outerbox.add(Box.createVerticalStrut(150));
                outerbox.add(Box.createHorizontalStrut(350));
                outerbox.validate();
                outerbox.repaint();
        }

        /**
         * Method to update the right side of the GUI --> the map.
         * @param map The new image to update the GUI with
         */
        public void updateRightSide(ImageIcon map){
                mapbox.removeAll();
                JPanel panel = new JPanel(new BorderLayout());
                JLabel lbl = new JLabel(map);
                panel.add(lbl, BorderLayout.WEST);
                mapbox.add(panel);
                mapbox.validate();
                mapbox.repaint();
        }

        /**
         * Method to add images to the GUI
         * @param url The url of the image
```

```
     * @return A JPanel with the image
     */
    public JPanel createImg(String url) {
            JPanel panel = new JPanel(new BorderLayout());
            ImageIcon icon = new ImageIcon(url);
            JLabel lbl = new JLabel(icon);
            panel.add(lbl, BorderLayout.WEST);
            return panel;
    }

    /**
     * Method to add text to the GUI
     * @param label The text one wants to add
     * @return A JPanel with the text
     */
    public JPanel createText(String label) {
            JPanel panel = new JPanel(new BorderLayout());
            JLabel lbltxt = new JLabel(label);
            panel.add(lbltxt, BorderLayout.WEST);
            return panel;
    }

    /**
     * The next methods and classes creates the combo
     * boxes that makes it possible to define the search
     * parameters
     * @return JPanels with the combo boxes
     */
    public JPanel createType () {
            JPanel panel = new JPanel(new BorderLayout());
            String[] typeStrings = { "Ingen valgt", "seng", "lege", "monitor", "tv" };
            JLabel lbl = new JLabel("Type:   ");
            ComboBoxType typeList = new ComboBoxType(typeStrings);
            panel.add(lbl, BorderLayout.WEST);
            panel.add(typeList, BorderLayout.EAST);
            return panel;
    }

    //inner class to create the ComboBox with listener
    public class ComboBoxType extends JPanel implements ActionListener {

            public ComboBoxType(String[] strings) {
                    super(new BorderLayout());
                    JComboBox list = new JComboBox(strings);
                    list.addActionListener(this);
                    add(list, BorderLayout.PAGE_START);
            }

            /** Listens to the combo box. */
            public void actionPerformed(ActionEvent e) {
                    JComboBox cb = (JComboBox)e.getSource();
                    type = (String)cb.getSelectedItem();
            }
    }

    public JPanel createAvdeling() {
            JPanel panel = new JPanel(new BorderLayout());
            String[] avdStrings = { "alle", "kreft", "barsel", "akutten" };
            JLabel lbl = new JLabel("Avdeling:   ");
            ComboBoxAvd avdList = new ComboBoxAvd(avdStrings);
```

```
        panel.add(lbl, BorderLayout.WEST);
        panel.add(avdList, BorderLayout.EAST);
        return panel;
}

public class ComboBoxAvd extends JPanel implements ActionListener {

        public ComboBoxAvd(String[] strings) {
                super(new BorderLayout());
                JComboBox list = new JComboBox(strings);
                list.addActionListener(this);
                add(list, BorderLayout.PAGE_START);
        }

        /** Listens to the combo box. */
        public void actionPerformed(ActionEvent e) {
                JComboBox cb = (JComboBox)e.getSource();
                avd = (String)cb.getSelectedItem();
        }
}

public JPanel createStatus() {
        JPanel panel = new JPanel(new BorderLayout());
        String[] statusStrings = { "alle", "ledig", "opptatt" };
        JLabel lbl = new JLabel("Status:   ");
        ComboBoxStatus statusList = new ComboBoxStatus(statusStrings);
        panel.add(lbl, BorderLayout.WEST);
        panel.add(statusList, BorderLayout.EAST);
        return panel;
}

public class ComboBoxStatus extends JPanel implements ActionListener {

        public ComboBoxStatus(String[] strings) {
                super(new BorderLayout());
                JComboBox list = new JComboBox(strings);
                list.addActionListener(this);
                add(list, BorderLayout.PAGE_START);
        }

        /** Listens to the combo box. */
        public void actionPerformed(ActionEvent e) {
                JComboBox cb = (JComboBox)e.getSource();
                status = (String)cb.getSelectedItem();
        }
}

/**
 * These two methods creates the text fields for the
 * search parameters 'navn' and 'unikID'.
 * @return JPanels with the text fields
 */
public JPanel createNavn() {
        JPanel panel = new JPanel(new BorderLayout());
        JLabel lbl = new JLabel("Navn:   ");
        navnTextField = new JTextField();
        navnTextField.setColumns(15);
        panel.add(lbl, BorderLayout.WEST);
        panel.add(navnTextField, BorderLayout.EAST);
```

```
        return panel;
    }

    public JPanel createUnik() {
        JPanel panel = new JPanel(new BorderLayout());
        JLabel lbl = new JLabel("Unik id:   ");
        unikTextField = new JTextField();
        unikTextField.setColumns(15);
        panel.add(lbl, BorderLayout.WEST);
        panel.add(unikTextField, BorderLayout.EAST);
        return panel;
    }

    /**
     * Method to create radio buttons
     * Used so that the actor can select objects from a list
     */
    public JPanel createRadioButton(String label) {
        JPanel panel = new JPanel(new BorderLayout());
        Radio theButton = new Radio(label);
        panel.add(theButton, BorderLayout.WEST);
        return panel;
    }

    public class Radio extends JPanel implements ActionListener {

        public Radio(String label) {
            super(new BorderLayout());
            JRadioButton aButton = new JRadioButton(label);
            group.add(aButton);
            aButton.addActionListener(this);
            add(aButton, BorderLayout.PAGE_START);
        }

        /** Listens to the radio button. */
        public void actionPerformed(ActionEvent e) {
            selected = (String)e.getActionCommand();
        }
    }

    /**---------------------Create Buttons----------------------------------------*/
    /**
     * The next methods creates all the different buttons that are
     * used in the GUI
     * @return JPanels with the buttons
     */
    public JPanel createSokButton() {
        JPanel panel = new JPanel(new BorderLayout());
        Action a = new AbstractAction("Søk") {
            public void actionPerformed(ActionEvent e) {
                searchForObjects();
            }
        };
        JButton b = createButton(a);
        b.setText("Søk");
        b.setMnemonic('S');
        panel.add(b, BorderLayout.WEST);
        return panel;
    }
```

```
public JButton createFindButton() {
        Action a = new AbstractAction("Finn") {
                public void actionPerformed(ActionEvent e) {
                        find();
                }
        };
        JButton b = createButton(a);
        b.setText("Finn");
        b.setMnemonic('F');
        return b;
}

public JButton createMonitorButton() {
        Action a = new AbstractAction("Monitor") {
                public void actionPerformed(ActionEvent e) {
                        monitor();
                }
        };
        JButton b = createButton(a);
        b.setText("Monitor");
        b.setMnemonic('M');
        return b;
}

public JButton createAlarmButton() {
        Action a = new AbstractAction("Alarm") {
                public void actionPerformed(ActionEvent e) {
                        alarm();
                }
        };
        JButton b = createButton(a);
        b.setText("Alarm");
        b.setMnemonic('A');
        return b;
}

public JPanel createBackButton() {
        JPanel panel = new JPanel(new BorderLayout());
        Action a = new AbstractAction("Nytt søk") {
                public void actionPerformed(ActionEvent e) {
                        back();
                }
        };
        JButton b = createButton(a);
        b.setText("Nytt søk");
        b.setMnemonic('N');
        panel.add(b, BorderLayout.WEST);
        return panel;
}

public JPanel createStopMonitorButton() {
        JPanel panel = new JPanel(new BorderLayout());
        Action a = new AbstractAction("Stopp monitor") {
                public void actionPerformed(ActionEvent e) {
                        stopMonitor();
                }
        };
        JButton b = createButton(a);
        b.setText("Stopp monitor");
        b.setMnemonic('T');
```

```
            panel.add(b, BorderLayout.WEST);
            return panel;
        }

        public JPanel createStopAlarmButton() {
            JPanel panel = new JPanel(new BorderLayout());
            Action a = new AbstractAction("Stopp alarm") {
                public void actionPerformed(ActionEvent e) {
                    stopAlarm();
                }
            };
            JButton b = createButton(a);
            b.setText("Stopp alarm");
            b.setMnemonic('L');
            panel.add(b, BorderLayout.WEST);
            return panel;
        }

        public JPanel createAckAlarmButton() {
            JPanel panel = new JPanel(new BorderLayout());
            Action a = new AbstractAction("OK") {
                public void actionPerformed(ActionEvent e) {
                    stopAlarm();
                }
            };
            JButton b = createButton(a);
            b.setText("OK");
            b.setMnemonic('O');
            panel.add(b, BorderLayout.WEST);
            return panel;
        }

        /**
         * All buttons are created using this method.
         * It specifies the appearance of the button
         * @param a The action connected to the current button
         * @return The JButton created
         */
        public JButton createButton(Action a) {
            JButton b = new JButton(a) {
                public Dimension getMaximumSize() {
                    int width = Short.MAX_VALUE;
                    int height = super.getMaximumSize().height;
                    return new Dimension(width, height);
                }
            };
            return b;
        }

        /**----------------Methods to send messages-----------------------*/

        /**
         * This method takes the parameters that the actor selected and sends
         * them to the DatabaseProxy.
         * This is sent as a message via the JavaFrame structure.
         */
        private void searchForObjects() {
            System.out.println("Input parameters from the user:");
            System.out.println("type= " + type + ", avd= " + avd + ", status= " + status + ",
navn= " + navnTextField.getText() + ", unik id= " + unikTextField.getText());
```

```
                System.out.println("");
                toTermAgent.input(new SearchForObjectsM(type, avd, status,
navnTextField.getText(), unikTextField.getText()));
        }

        /**
         * This method takes the coordinates of the selected object
         * and sends them to the MapProxy to get a map.
         * This is sent as a message via the JavaFrame structure.
         */
        public void find(){
                System.out.println("Terminal: want to view the location of : " + selected);

                //To send a message :
                String xcoord = "";
                String ycoord = "";

                for(int i=0; i<result.size(); i++){
                        String all = (String) result.get(i);
                        StringTokenizer st = new StringTokenizer(all, ",");
                        String aname = st.nextToken();
                        if(aname.equals(selected)){
                                String nothing = st.nextToken();
                                xcoord = st.nextToken();
                                ycoord = st.nextToken();
                        }//if
                }//for
                toTermAgent.input(new ShowMapM(xcoord, ycoord));
        }

        /**
         * This method takes the uID of the selected object and
         * sends a request to the Location Server to start
         * TRACK'in this object.
         * This is sent as a message via the JavaFrame structure.
         */
        public void monitor(){
                String uID = "";

                for(int i=0; i<result.size(); i++){
                        String all = (String) result.get(i);
                        StringTokenizer st = new StringTokenizer(all, ",");
                        String aname = st.nextToken();
                        if(aname.equals(selected)){
                                uID = st.nextToken();
                        }//if
                }//for

                System.out.println("Terminal: want to monitor  : " + selected + ". Unik ID er :
" + uID);
                toTermAgent.input(new MonitorObjectsM(uID));

                left = false;
                updateLeftSide("monitorer");
        }

        /**
         * This method takes the uID of the selected object and
         * sends a request to the Location Server to set a
         * NOTIFY message on the object.
```

```
         * This is sent as a message via the JavaFrame structure.
         */
        public void alarm(){
                String uID = "";
                String xcoord = "";
                String ycoord = "";

                for(int i=0; i<result.size(); i++){
                        String all = (String) result.get(i);
                        StringTokenizer st = new StringTokenizer(all, ",");
                        String aname = st.nextToken();
                        if(aname.equals(selected)){
                                uID = st.nextToken();
                                xcoord = st.nextToken();
                                ycoord = st.nextToken();
                        }//if
                }//for

                System.out.println("Terminal: want to get alarm for : " + selected + ". Unik ID
er : " + uID);
                toTermAgent.input(new StartAlarmM(uID));

                left = false;
                updateLeftSide("alarmer");

                //to show the original location of object:
                toTermAgent.input(new ShowMapM(xcoord, ycoord));

        }

        /**
         * Method to go back to start a new search
         *
         */
        public void back(){
                System.out.println("vi går tilbake...");
                updateLeftSide("sok");
                createMap();
        }

        /**
         * Method to stop monitoring an object.
         * Sends a request to the Location Server to send
         * a DELETE message for this object.
         */
        public void stopMonitor(){
                System.out.println("");
                System.out.println("Stopping monitor!");
                toTermAgent.input(new EndMonitorM());
                updateLeftSide("sok");
                createMap();
        }

        /**
         * Method to stop alarm for an object.
         * Sends a request to the Location Server to send
         * a DELETE message for this object.
         */
        public void stopAlarm(){
                System.out.println("");
```

```
                System.out.println("Stopping the alarm!");
                toTermAgent.input(new EndAlarmM());
                updateLeftSide("sok");
                createMap();
        }

        /**----------------Methods when receiving messages-------------------------*/

        /**
         * Method that receives the result after a search for objects
         * has been performed.
         * Updates the left side of the GUI.
         */
        public void receiveSearchResult(ArrayList result){
                this.result = result;
                System.out.println("**Terminal: ResultList received: ");
                System.out.println("-----------------------");
                for(int i=0; i<result.size(); i++){
                        System.out.println(result.get(i));
                        System.out.println("-----------------------");
                }
                System.out.println("");
                System.out.println("");
                System.out.println("");
                updateLeftSide("resultat");
        }

        /**
         * Method that receives the result after a call to the
         * Map Server has been performed.
         * Updates the right side of the GUI with the new map.
         * @param map The image returned from the call
         */
        public void receiveFindResult(ImageIcon map){
                System.out.println("**Terminal: Map received.. ");
                if(left){
                        updateLeftSide("finn");
                }
                updateRightSide(map);
                System.out.println("");
                System.out.println("");
                System.out.println("");
        }

        /**
         * Method that receives new alarms.
         * Updates the left side of the GUI with an alarm message.
         */
        public void receiveAlarmResult(){
                System.out.println("**Terminal: New alarm message received!");
                updateLeftSide("alarmNow");
                System.out.println("");
        }

}//class
```

```
/*************************************************************
 * This is the edge mediator from the from Terminal to TerminalAgent
 *
 * EdgeInTerm.java
 *
 * @author Solrun Pedersen
 *************************************************************/

package com.controll;
import se.ericsson.eto.norarc.javaframe.*;
import com.messages.*;

public class EdgeInTerm extends SimpleRouterMediator {

        ExMain main;

        /**
         * Constructor
         * @param main A reference to the main class
         */
        public EdgeInTerm(ExMain main) {
        super();
        this.main = main;
        System.out.println("**EdgeInTerm created");
        }

        /**
         * Method to forward received messages
         */
        public void forward(Message sig){
                Mediator next;

                if(sig instanceof SearchForObjectsM ){
                next = (Mediator) mediatorList.get(0);
                        System.out.println("**EdgeInTerm: Sendt: " + sig.messageName() +
sig.messageContent());
                next.forward( sig );
                }
                else if(sig instanceof ShowMapM ){
                        next = (Mediator) mediatorList.get(0);
                        System.out.println("**EdgeInTerm: Sendt: " + sig.messageName() +
sig.messageContent());
                        next.forward( sig );
                }
                else if(sig instanceof MonitorObjectsM ){
                        next = (Mediator) mediatorList.get(0);
                        System.out.println("**EdgeInTerm: Sendt: " + sig.messageName() +
sig.messageContent());
                        next.forward( sig );
                }
                else if(sig instanceof EndMonitorM ){
                        next = (Mediator) mediatorList.get(0);
                        System.out.println("**EdgeInTerm: Sendt: " + sig.messageName() +
sig.messageContent());
                        next.forward( sig );
                }
```

```
                else if(sig instanceof StartAlarmM ){
                        next = (Mediator) mediatorList.get(0);
                        System.out.println("**EdgeInTerm: Sendt: " + sig.messageName() +
sig.messageContent());
                        next.forward( sig );
                }
                else if(sig instanceof EndAlarmM ){
                        next = (Mediator) mediatorList.get(0);
                        System.out.println("**EdgeInTerm: Sendt: " + sig.messageName() +
sig.messageContent());
                        next.forward( sig );
                }
        }//forward

}//class
```

```
/*************************************************************
 * This is the edge mediator from the from TerminalAgent to Terminal
 *
 * EdgeOutTerm.java
 *
 * @author Solrun Pedersen
 *************************************************************/

package com.controll;

import se.ericsson.eto.norarc.javaframe.*;
import com.messages.*;


public class EdgeOutTerm extends Mediator {

        ExMain main;

        /**
         * Constructor
         * @param main A reference to the main class
         */
        public EdgeOutTerm(ExMain main) {
        super();
        this.main = main;
        System.out.println("**EdgeOutTerm created");
        }

        /**
         * Method to forward received messages
         */
        public void forward(Message sig) {

        if(sig instanceof ResultListM){
                System.out.println("**EdgeOutTerm: Sendt: " + sig.messageName()+
sig.messageContent());
                        ResultListM im = (ResultListM)sig;
                        main.terminal.receiveSearchResult(im.result);
        }
        else if(sig instanceof MapWithObjectsM){
                        System.out.println("**EdgeOutTerm: Sendt: " + sig.messageName()+
sig.messageContent());
```

```
                        MapWithObjectsM im = (MapWithObjectsM)sig;
                        main.terminal.receiveFindResult(im.map);
        }
                else if(sig instanceof AlarmToTerminalM){
                        System.out.println("**EdgeOutTerm: Sendt: " + sig.messageName() +
sig.messageContent());
                        main.terminal.receiveAlarmResult();
                }
        else {
                System.out.println("**Error in EdgeOutTerm: unexpected signal type received**");
//Hvis det er et ugyldig signal
                }
        }
}
```

```
/*************************************************************
 * A class that translates between JavaFrame and the
 * Object Database.
 *
 * DatabaseProxy.java
 *
 * @author Solrun Pedersen
 *************************************************************/

package com.controll;

import se.ericsson.eto.norarc.javaframe.*;
import com.messages.*;

import java.sql.Connection;
import java.sql.SQLException;
import java.util.ArrayList;
import javax.naming.NamingException;
import no.geomatikk.database.Database;
import no.geomatikk.database.newpoi.Attribute;
import no.geomatikk.database.newpoi.Location;
import no.geomatikk.database.newpoi.Poi;
import no.geomatikk.database.newpoi.StoredProceduresPoi;
import oracle.sdoapi.adapter.GeometryInputTypeNotSupportedException;
import oracle.sdoapi.geom.CoordPoint;
import oracle.sdoapi.geom.Geometry;
import oracle.sdoapi.geom.InvalidGeometryException;


public class DatabaseProxy {

        Mediator toRouter;

        Database db = new Database();
    private Connection myConn;
    private StoredProceduresPoi mySP;

        String type;
        String avdeling;
        String status;
        String navn;
        String unikId;
        ArrayList result;
```

```
        Poi[] testPois = null;
        Poi testPoi = null;
        Poi[] avdelinger = null;

        int type2 = 0;
        int subType = 0;
        String name = null;
        String theName = null;
        boolean foundName = false;
        Long avdelingUid = null;
        boolean doc = false;
        boolean unik = false;

        /**
         * Constructor.
         * @param toS The input mediator from the DatabaseProxy
         * to the JavaFrame system
         */
        public DatabaseProxy(Mediator toS){
                System.out.println("**DatabaseProxy: New DatabaseProxy created.");
                toRouter  = toS;
                result = new ArrayList();
        }

        /**
         * Creates the connection to the Object Database
         * @return The connection
         */
        private Connection getConnection() throws NamingException, SQLException {
                return Database.getConnection("jdbc:oracle:thin:@10.254.10.142:1521:brapt",
"brapt", "brapt");
        }

        /**
         * Method to search for objects matching certain parameters input from the user.
         * @param type The type of the object
         * @param avdeling The ward that the object belongs to
         * @param status The current status of the object
         * @param navn Used when searching for persons. The name of the person
         * @param unikId The unique identification number of the object
         */
        public void search(String type, String avdeling, String status, String navn, String
unikId) throws SQLException, InvalidGeometryException, GeometryInputTypeNotSupportedException{
                try {
                        myConn = getConnection();
                        mySP = new StoredProceduresPoi(myConn);
                } catch (Exception e) {
                        e.printStackTrace();
                }
                this.type = type;
                this.avdeling = avdeling;
                this.status = status;
                this.navn= navn;
                this.unikId=unikId;

                if(status.equals("alle")){
                        //Search on every poi, no matter what the 'status' is.
                        status = null;
                }
```

```
                //Finds the 'avdeling' that matches the search.
                if(!avdeling.equals("alle")){
                        avdelinger = mySP.getPois("100100", avdeling, null, "IGIS", 4, 1, null,
null, true, false, false, true, false, false, true, false, true, false, false);
                        if (avdelinger != null) {
                           for (int i=0; i<avdelinger.length; i++) {
                                        avdelingUid =avdelinger[i].getUid();
                           }
                        }
                }
                //Unik id has 1. priortiy (is the field searched first).
                if(!unikId.equals("")) {
                 try {
                        testPoi = mySP.getPoi(new Long(unikId), true, false, false, true, false,
false, true, false, false, true, false);
                        if (testPoi != null) {
                           unik = true;
                           if(testPoi.getTitle().startsWith("Lege")){
                                  doc = true;

                                }//if testPoi.getT
                        }//if testPoi null
                 } catch (Exception e) {
                         e.printStackTrace();
                 }
                } //end if

                //Name has 2. priority (is the one searched if 'unikId' is not choosen).
                else if(!navn.equals("")){
                doc = true;
                testPois = mySP.getPois("100100", "Lege%", null, "IGIS", 2, 1, null, null, true,
false, false,true, false, false, true, false, true, false, false);
                if (testPois != null) {
                        for (int i=0; i<testPois.length; i++) {
                                Attribute[] myAttributes = testPois[i].getAttributes();
                                if(myAttributes != null){
                                        for (int j=0; j<myAttributes.length; j++) {
                                                name = myAttributes[j].getValue();
                                                String n = navn.toLowerCase();
                                                String m = name.toLowerCase();
                                                if(m.equals(n) || m.startsWith(n)){
                                                        foundName = true;
                                                        theName = name;
                                                        testPoi = testPois[i];
                                                }//if
                                        }//for
                                } //if
                         }//for
                 }//if
          } //else if

          //If not 'unikId' or 'navn' is choosen, search on 'type'.
          else {
                if(type.equals("tv")){
                  type2 = 1;
                  subType = 1;
                } else if(type.equals("monitor")){
                  type2 = 1;
                  subType = 2;
                } else if(type.equals("lege")){
```

```
                type2 = 2;
                subType = 1;
               //You cannot reserve a doctor!
                status = null;
               //If a doctor: display his name, instead of title (ex: doc1).
                doc = true;
              } else if(type.equals("seng")){
                type2 = 2;
                subType = 2;
              }
              testPois = mySP.getPois("100100", type + "%", status, "IGIS", type2, subType,
null, null, true, false, false, true, false, false, true, false, true, false, false);
        }//end else

       String valg = null;
       boolean noMatch = false;
       int numberWrite = 0;

       if (testPois != null) {
      if(foundName){
               String loc  = getLocation(testPoi);
               if(!loc.equals("none")){
                     String element = theName + "," + testPoi.getUid() + "," + loc;
                     result.add(element);
                     numberWrite = numberWrite + 1;
               }
         } //if foundName
         else{
                for (int i=0; i<testPois.length; i++) {

                      if(!avdeling.equals("alle")){
                            Long[] parents = testPois[i].getParents();
                            if(parents != null){
                                  for (int j=0; j<parents.length; j++) {
                                        if(parents[j].equals(avdelingUid)){
                                              if(doc){
                                                    String docName = null;
                                                    Attribute[] myAttributes =
testPois[i].getAttributes();

                                                    if(myAttributes != null){
                                                          for (int m=0;

m<myAttributes.length; m++) {

                                                                docName =

myAttributes[m].getValue();

                                                          }//for
                                                    } //if
                                                    String loc  =

getLocation(testPois[i]);

                                                    if(!loc.equals("none")){
                                                          String element =

docName + "," + testPois[i].getUid() + "," + loc;

result.add(element);

                                                          numberWrite =

numberWrite + 1;

                                                    }
                                              }//if doc
                                              else {
                                                   String loc  =

getLocation(testPois[i]);

                                                    if(!loc.equals("none")){
```

```
                                                                    String element =
testPois[i].getTitle() + "," + testPois[i].getUid() + "," + loc;
                                                                    result.add(element);
                                                                    numberWrite = numberWrite
+ 1;
                                                                }
                                                            }//else
                                                        }//if parents
                                                    }//for
                                                }//if parents
                                            } //if avdeling
                                            else{
                                                    //'Avdeling' not choosen
                                            if(doc){
                                                    String docName = null;
                                                    Attribute[] myAttributes = testPois[i].getAttributes();
                                                        if(myAttributes != null){
                                                            for (int m=0; m<myAttributes.length; m++) {
                                                                docName = myAttributes[m].getValue();
                                                            }//for
                                                        } //if
                                                    String loc  = getLocation(testPois[i]);
                                                        if(!loc.equals("none")){
                                                            String element = docName + "," +
testPois[i].getUid() + "," + loc;
                                                            result.add(element);
                                                            numberWrite = numberWrite + 1;
                                                        }
                                                    }//if doc
                                                    else {
                                                        String loc  = getLocation(testPois[i]);
                                                          if(!loc.equals("none")){
                                                            String element = testPois[i].getTitle() + "," +
testPois[i].getUid() + "," + loc;
                                                            result.add(element);
                                                            numberWrite = numberWrite + 1;
                                                        }
                                                    }//else
                                            }//not 'avdeling'
                                    }//for hele
                            }//else
                            } else {
                                if(unik){
                                    //'UnikId' is choosen.
                                        if(doc){
                                            String docName = null;
                                            Attribute[] myAttributes = testPoi.getAttributes();
                                            if(myAttributes != null){
                                                for (int m=0; m<myAttributes.length; m++) {
                                                    docName = myAttributes[m].getValue();
                                                }//for
                                            } //if
                                            String loc  = getLocation(testPoi);
                                            if(!loc.equals("none")){
                                                String element = docName + "," + testPoi.getUid() + "," + loc;
                                                result.add(element);
                                                numberWrite = numberWrite + 1;
                                            }
                                        }//if doc
                                        else {
                                            String loc  = getLocation(testPoi);
```

```
                        if(!loc.equals("none")){
                                String element = testPoi.getTitle() + "," + testPoi.getUid() +
"," + loc;
                                result.add(element);
                                numberWrite = numberWrite + 1;
                        }
                 }//else
           }//if unik
        }//end else if
        if(numberWrite == 0){
                String element = "Ingen match funnet til ditt søk. Vennligst gå tilbake og prøv
igjen";
                result.add(element);
                //noMatch = true;
        }//end if (numberWrite)

        System.out.println("**DatabaseProxy: Sendt: ResultListM");
        //sends result message back:
        toRouter.input(new ResultListM(result));
   }//end search method

        /**
         * This method is used to retrieve the current location
         * of an object from the Object Database
         * @param poi The object to retrieve the location for
         * @return The location in x-, y- and z-coordinates
         */
   public String getLocation(Poi poi){

         double x = 0.0;
         double y = 0.0;
         double z = 0.0;
         int eta = 0;
         String location = "none";
         Location[] locations = poi.getLocations();
         //Find the most recent location:
         int locationId = 0;
         int locPosition = 0;
         if(locations != null){
                for(int j=0;j<locations.length;j++){
                        int loc = (locations[j].getLocationId()).intValue();
                        if(loc > locationId){
                                locationId = loc;
                                locPosition = j;
                        }
                }
          }
         Geometry g= locations[locPosition].getGeometry();
         CoordPoint p = g.getLabelPoint();
         x = p.getX();
         y = p.getY();
         z = p.getZ();
         //the z-value is for selecting the floor.

        //Return only the POI's on the first floor for convenience in this demonstrator...
         if(z == 1.0){
             location = x + "," + y;
         }
                return location;
   }//end getLocation
```

```
}//end class
```

```
/*************************************************************
 * This is the edge mediator from the from DatabaseProxy to Router
 *
 * EdgeInDb.java
 *
 * @author Solrun Pedersen
 *************************************************************/

package com.controll;
import se.ericsson.eto.norarc.javaframe.*;
import com.messages.*;

public class EdgeInDb extends SimpleRouterMediator {

        ExMain main;

        /**
         * Constructor
         * @param main A reference to the main class
         */
        public EdgeInDb(ExMain main) {
                super();
                this.main = main;
                System.out.println("**EdgeInDb created");
        }

        /**
         * Method to forward received messages
         */
        public void forward(Message sig){
                Mediator next;

                if(sig instanceof ResultListM ){
                        next = (Mediator) mediatorList.get(0);
                        next.forward( sig );
                        System.out.println("**EdgeInDb: Sendt: " + sig.messageName() +
sig.messageContent());
                }
        }//forward

}//class
```

```
/*************************************************************
 * This is the edge mediator from the from Router to DatabaseProxy
 *
 * EdgeOutDb.java
 *
 * @author Solrun Pedersen
 *************************************************************/

package com.controll;

import se.ericsson.eto.norarc.javaframe.*;
```

```
import com.messages.*;

public class EdgeOutDb extends Mediator {

	ExMain main;

	/**
	 * Constructor
	 * @param main A reference to the main class
	 */
	public EdgeOutDb(ExMain main) {
	super();
	this.main = main;
	System.out.println("**EdgeOutDb created");
	}

	/**
	 * Method to forward received messages
	 */
	public void forward(Message sig) {

	if(sig instanceof SearchForObjectsM){
			SearchForObjectsM im = (SearchForObjectsM)sig;
			System.out.println("**EdgeOutDb: Sendt: " + sig.messageName() +
sig.messageContent());
			System.out.println("**EdgeOutDb: parameters: " + im.type + im.avd +
im.status + im.navn + im.unikId);
			System.out.println("");
			try {
				main.dbp.search(im.type, im.avd, im.status, im.navn, im.unikId);
			} catch (Exception e) {
				e.printStackTrace();
			}
		}
	else{
		System.out.println("**Error i EdgeOutDb: unexpected signal type from System to
database"); //Hvis det er et ugyldig signal
		}
	}
}
```

```
/*************************************************************
 * A class that translates between JavaFrame and the
 * Map Server.
 *
 * MapProxy.java
 *
 * @author Solrun Pedersen
 *************************************************************/

package com.controll;

import se.ericsson.eto.norarc.javaframe.*;
import com.messages.*;

import javax.swing.*;
import java.awt.*;
import java.io.*;
```

```
import java.net.*;

public class MapProxy extends JPanel{

	Mediator toRouterFromMap;

	String baseURL = "http://demo.geomatikk.no/servlet/MapProvider?";
	String cx = "CX=";
	String cy = "&CY=";
	String rest =
"&ACTION=point&IW=550&IH=550&SC=500&MAP=864&MAPUSER=Areal&MAPPASSWORD=Areal01";

	String xcoord;
	String ycoord;

	/**
	 * Constructor.
	 * @param toS The input mediator from the MapProxy
	 * to the JavaFrame system
	 */
	public MapProxy(Mediator toS){
		System.out.println("**MapProxy: New MapProxy created.");
		toRouterFromMap = toS;
	}

	/**
	 * This method retreives a map with one object drawn in from the Map System via a HTTP
call.
	 * The result is translated to an ImageIcon, and sent as a message to the Terminal
	 * via the JavaFrame structure.
	 * @param xcoord The x coordinate for the object
	 * @param ycoord The y coordinate for the object
	 */
	public void getMap(String xcoord, String ycoord){
		this.xcoord = xcoord;
		this.ycoord = ycoord;
		System.out.println("**MapProxy: Received "  + xcoord + "," + ycoord);

		String url = baseURL + cx + xcoord + cy + ycoord + rest;

		try {
			URL queryURL = new URL(url);
			System.out.println("**MapProxy: Map url is called: " + queryURL);

			//Get map from web-server:
			HttpURLConnection connection = (HttpURLConnection)
queryURL.openConnection();
			ByteArrayOutputStream byteStream = new ByteArrayOutputStream();
			InputStream httpStream = connection.getInputStream();
			int count;
			while ((count = httpStream.read()) != -1) {
				byteStream.write(count);
			}
			httpStream.close();
			byteStream.close();

			// Create image
			byte[] image_bytes = byteStream.toByteArray();
			System.out.println("");
```

```
                        System.out.println("**MapProxy: Got a map. Loaded " + image_bytes.length
+ " bytes from URL: " + queryURL.toString());
                        Image image = Toolkit.getDefaultToolkit().createImage(image_bytes);
                        MediaTracker mt = new MediaTracker(this);
                        mt.addImage(image, 0);
                        try {
                                mt.waitForAll();
                        } catch (InterruptedException ie) {
                               ie.printStackTrace();
                               System.out.println("something went wrong... ");
                        }
                        ImageIcon icon = new ImageIcon(image);

                        //sends result message back:
                        System.out.println("**MapProxy: Sendt: MapWithObjectsM");
                        toRouterFromMap.input(new MapWithObjectsM(icon));

                } catch (Exception e) {
                        e.printStackTrace();
                }
        }//end method getMap

}
```

```
/*************************************************************
 * This is the edge mediator from the from MapProxy to Router
 *
 * EdgeInMap.java
 *
 * @author Solrun Pedersen
 *************************************************************/

package com.controll;
import se.ericsson.eto.norarc.javaframe.*;
import com.messages.*;

public class EdgeInMap extends SimpleRouterMediator {

        ExMain main;

        /**
         * Constructor
     * @param main A reference to the main class
     */
        public EdgeInMap(ExMain main) {
                super();
                this.main = main;
                System.out.println("**EdgeInMap created");
        }

        /**
         * Method to forward received messages
     */
        public void forward(Message sig){
                Mediator next;

                if(sig instanceof MapWithObjectsM ){
```

```
                        next = (Mediator) mediatorList.get(0);
                        next.forward( sig );
                        System.out.println("**EdgeInMap: Sendt: " + sig.messageName() +
sig.messageContent());
                }
        }//forward

}//class
```

```
/*************************************************************
 * This is the edge mediator from the from Router to MapProxy
 *
 * EdgeOutMap.java
 *
 * @author Solrun Pedersen
 *************************************************************/

package com.controll;

import se.ericsson.eto.norarc.javaframe.*;
import com.messages.*;

public class EdgeOutMap extends Mediator {

        ExMain main;

        /**
         * Constructor
         * @param main A reference to the main class
         */
        public EdgeOutMap(ExMain main) {
        super();
        this.main = main;
        System.out.println("**EdgeOutMap created");
        }

        /**
         * Method to forward received messages
         */
        public void forward(Message sig) {

                if(sig instanceof ShowMapM){
                        ShowMapM im = (ShowMapM)sig;
                        System.out.println("**EdgeOutMap: Sendt: " + sig.messageName() +
sig.messageContent());
                        System.out.println("**EdgeOutMap: parameters: xcoord=" + im.xcoord +"
ycoord=" + im.ycoord);
                        System.out.println("");
                        main.mp.getMap(im.xcoord, im.ycoord);
                }
        else{
                System.out.println("**Error i EdgeOutMap: unexpected signal type from System to
database"); //Hvis det er et ugyldig signal
                }
        }

}
```

```
/*************************************************************
 * A class that translates between JavaFrame and the
 * Location Server.
 *
 * LocServProxy.java
 *
 * @author Solrun Pedersen
 *************************************************************/

package com.controll;

import se.ericsson.eto.norarc.javaframe.*;
import com.messages.*;
import com.telenor.positionserver.*;

import java.util.Hashtable;
import javax.jms.*;
import javax.naming.Context;
import javax.naming.InitialContext;

public class LocServProxy{

	Mediator toLocAgent;
	String unikID;
	String action;

	QueueConnection connection = null;
	QueueSession session = null;
	TemporaryQueue replyQueue = null;
	static QueueConnectionFactory factory = null;
	static Context context = null;
	static String queue_name = "position";
	static int ackMode = Session.AUTO_ACKNOWLEDGE;

	PosRequest u = null;
	ObjectMessage message = null;
	QueueSender sender = null;

	/**
	 * Constructor
	 * @param toLocAgent The input mediator from the LocServProxy
	 * to the JavaFrame system
	 */
	public LocServProxy(Mediator toLocAgent){
		System.out.println("**LocServProxy: New LocServProxy created.");
		this.toLocAgent = toLocAgent;

	}

	/**
	 * Method to send a message to the Location Server.
	 * Used to start TRACK or NOTIFY for an object.
	 * @param unikID The unique identification number of the object
	 * @param action Decides which message to send (TRACK or NOTIFY)
	 */
	public void sendMessage(String unikID, String action) {
```

```
                this.unikID = unikID;
                this.action = action;
                System.out.println("**LocServProxy: Received: unikID : "  + unikID + ". Action =
" + action);
                try {
                        Hashtable props = new Hashtable();
                        props.put(Context.PROVIDER_URL, "tcp://129.241.219.152:3035/");
                        props.put(Context.INITIAL_CONTEXT_FACTORY,
"org.exolab.jms.jndi.mipc.IpcJndiInitialContextFactory");
                        context = new InitialContext(props);

                        // lookup the connection factory from the context
                        factory = (QueueConnectionFactory)
context.lookup("JmsQueueConnectionFactory");
                        // if we can't find the factory then throw an exception
                        if (factory == null) {
                                throw new RuntimeException("**LocServProxy: Failed to locate
connection factory");
                        }
                        connection = factory.createQueueConnection();
                        connection.start();
                        session = connection.createQueueSession(false,
Session.AUTO_ACKNOWLEDGE);
                        Queue queue = (Queue) context.lookup("position");
                        if (queue == null) {
                                throw new RuntimeException("**LocServProxy: Failed to create the
queue " + queue);
                        }
                        sender = session.createSender(queue);
                        replyQueue = session.createTemporaryQueue();

                        String[] devices = {"urn:test:object"};

                        if(action.equals("monitor")){
                                u = new PosRequest("ingebrigt", "test", PosRequest.TRACK,
devices);
                                message = session.createObjectMessage(u);
                                message.setJMSReplyTo(replyQueue);

                                // .. and send
                                sender.send(message, DeliveryMode.PERSISTENT, 2, 0);
                                System.out.println("**LocServProxy: TRACK request sendt");
                        }
                        else if(action.equals("alarm")){
                                u = new PosRequest("ingebrigt", "test", PosRequest.NOTIFY,
devices);
                                NotifyArea na = new NotifyArea(63.422366770019496f,
10.4379575192261f, 25);
                                na.setInside(false); //NOTIFY when _outside_ area
                                u.setNotifyArea(na);
                                message = session.createObjectMessage(u);
                                message.setJMSReplyTo(replyQueue);

                                //.. and send
                                sender.send(message, DeliveryMode.PERSISTENT, 2, 0);
                                System.out.println("**LocServProxy: NOTIFY request sendt");
                        }

                        // Start reply thread
                        Reply reply = new Reply();
```

```
                        reply.start();

                } catch (Exception exception) {
                        exception.printStackTrace();
                }
        }

        /**
         * This method is used to stop monitoring of an object, or to
         * stop getting alarm for an object.
         * Sends a DELETE message to the Location Server.
         * @param what Tells which action the DELETE message is for
         */
        public void end(String what){

                u.setRequestType(PosRequest.DELETE);
                try {
                        message = session.createObjectMessage(u);
                        message.setJMSReplyTo(replyQueue);
                        sender.send(message, DeliveryMode.PERSISTENT, 2, 0);

                        // Finished
                        sender.close();
                } catch (JMSException e) {
                        e.printStackTrace();
                }
                if(what.equals("monitor")){
                        System.out.println("**LocServProxy: Received: EndMonitorM. Sending
DELETE to Location Server..");
                }
                else if(what.equals("alarm")){
                        System.out.println("**LocServProxy: Received: EndAlarmM. Sending DELETE
to Location Server..");
                }
                System.out.println("");
                System.out.println("");
        }

/**
 * A class that creates the Reply Thread - a thread that
 * receives replies from the positionserver via JMS.
 */
public class Reply extends Thread implements ExceptionListener {

        /**
         * Method to receive the replies.
         * When a reply is received, a message is sent back to the Terminal
         * via the JavaFrame structure.
         */
        public void run() {
                System.out.println("**LocServProxy: Waiting for result");
                try {
                        while (true) {
                                QueueReceiver receiver = session.createReceiver(replyQueue);
                                ObjectMessage message = (ObjectMessage) receiver.receive();
                                Position position = (Position) message.getObject();
                                System.out.println("\n");
                                System.out.println("**LocServProxy: Received position = " +
position.toString());
                                double x = (double)position.getLongitude();
```

```
                                System.out.println("x = " + x);
                                String xcoord = "" + x;
                                double y = (double)position.getLatitude();
                                System.out.println("y = " + y);
                                String ycoord = "" + y;
                                //double z = (double)position.getAltitude();
                                double z = 1.0;
                                System.out.println("z = " + z);
                                if(action.equals("monitor")){
                                        System.out.println("**LocServProxy: Sendt: NewLocationM.
xcoord = " + xcoord + " ycoord = " + ycoord);
                                        System.out.println("");
                                        toLocAgent.input(new NewLocationM(xcoord, ycoord));
                                }
                                else if(action.equals("alarm")){
                                        System.out.println("**LocServProxy: Sendt: xcoord = " +
xcoord + " ycoord = " + ycoord);
                                        System.out.println("");
                                        toLocAgent.input(new AlarmM(xcoord, ycoord));
                                }

                        }//while(true)
                        } catch (Exception ee) {
                                System.out.println("**LocServProxy: Unexpected exception
reply(): " + ee.toString());
                                ee.printStackTrace();
                        } finally {
                                try {
                                        if (session != null)
                                                session.close();
                                        if (connection != null)
                                                connection.close();
                                } catch (JMSException e) {
                                        e.printStackTrace();
                                }
                        }
                }

                // implementation of ExceptionListener.onException
                public void onException(JMSException exception) {
                        try {
                                connection.close();
                        } catch (Exception error) {
                                error.printStackTrace();
                        }
                        System.out.println("**LocServProxy: Exit signal from user");
                        System.exit(0);
                }
        }
}
```

```
/*********************************************************
 * This is the edge mediator from the from Location Server to LocationAgent
 *
 * EdgeInLocServ.java
 *
 * @author Solrun Pedersen
 *********************************************************/
```

```
package com.controll;
import se.ericsson.eto.norarc.javaframe.*;
import com.messages.*;

public class EdgeInLocServ extends SimpleRouterMediator {

        ExMain main;

        /**
         * Constructor
     * @param main A reference to the main class
     */
        public EdgeInLocServ(ExMain main) {
        super();
        this.main = main;
        System.out.println("**EdgeInLocServ created");
        }

        /**
         * Method to forward received messages
         */
        public void forward(Message sig){
                Mediator next;

                if(sig instanceof NewLocationM ){
                next = (Mediator) mediatorList.get(0);
                        System.out.println("**EdgeInLocServ: Sendt: " + sig.messageName() +
sig.messageContent());
                next.forward( sig );
                }
                else if(sig instanceof AlarmM ){
                        next = (Mediator) mediatorList.get(0);
                        System.out.println("**EdgeInLocServ: Sendt: " + sig.messageName() +
sig.messageContent());
                        next.forward( sig );
                }
        }//forward

}//class
```

```
/*************************************************************
 * This is the edge mediator from the from LocationAgent to Location Server
 *
 * EdgeOutLocServ.java
 *
 * @author Solrun Pedersen
 *************************************************************/

package com.controll;

import se.ericsson.eto.norarc.javaframe.*;
import com.messages.*;

public class EdgeOutLocServ extends Mediator {
```

```
        ExMain main;

        /**
         * Constructor
         * @param main A reference to the main class
         */
        public EdgeOutLocServ(ExMain main) {
        super();
        this.main = main;
        System.out.println("**EdgeOutLocServ created");
        }

        /**
         * Method to forward received messages
         */
        public void forward(Message sig) {

        if(sig instanceof MonitorObjectsM){
                        MonitorObjectsM im = (MonitorObjectsM)sig;
                        System.out.println("**EdgeOutLocServ: Sendt: " + sig.messageName() +
sig.messageContent());
                        System.out.println("**EdgeOutLocServ: parameters: uId = " + im.unikID);
                        System.out.println("");
                        String action  = "monitor";
                main.lsp.sendMessage(im.unikID, action);
        }
                else if(sig instanceof EndMonitorM){
                        System.out.println("**EdgeOutLocServ: Sendt: " + sig.messageName()+
sig.messageContent());
                        String action  = "monitor";
                        main.lsp.end(action);
                }
                else if(sig instanceof StartAlarmM){
                        StartAlarmM im = (StartAlarmM)sig;
                        System.out.println("**EdgeOutLocServ: Sendt: " + sig.messageName() +
sig.messageContent());
                        System.out.println("**EdgeOutLocServ: parameters: uId = " + im.unikID);
                        System.out.println("");
                        String action  = "alarm";
                        main.lsp.sendMessage(im.unikID, action);
                }
                else if(sig instanceof EndAlarmM){
                        System.out.println("**EdgeOutLocServ: Sendt: " + sig.messageName()+
sig.messageContent());
                        String action  = "alarm";
                        main.lsp.end(action);
                }
                else{
        System.out.println("**Error i EdgeOutLocServ: unexpected signal type from System to
database"); //Hvis det er et ugyldig signal
                }
        }
}
```

G.2 package com.terminal

```
/*************************************************************
 * This is the State Machine for the TerminalAgent.
 * It starts the execution.
 *
 * TerminalAgentSM.java
 *
 * @author Solrun Pedersen
 *************************************************************/

package com.terminal;
import se.ericsson.eto.norarc.javaframe.*;

public class TerminalAgentSM extends StateMachine {

        static CompositeState states = new TerminalAgentCS("UserClient states");


        // Declare local references to output mediators
        protected Mediator TaToEnv;
        protected Mediator TaToRout;

        /**
         * Constructor.
         * @param sched Sheduler for the system
         * @param TaToEnv Mediator from the TerminalAgent to the Terminal
         * @param TaToRout Mediator from the TerminalAgent to the Router
         * @param TaToSelf
         */
        public TerminalAgentSM(Scheduler sched, Mediator TaToEnv, Mediator TaToRout, Mediator
TaToSelf) {
                super(sched);
                this.TaToEnv = TaToEnv;
                this.TaToRout = TaToRout;
                TaToSelf.addAddress(this);
        }

        /**
         * Method to start the Transition of the state machine
         */
        protected void execStartTransition() {
                states.enterState(this);
        }
}
```

```
/*************************************************************
 * This is the Composite State for the TerminalAgent.
 * It declares all the states, and which messages it can receive
 * and send in the different states.
 *
 * TerminalAgentCS.java
 *
 * @author Solrun Pedersen
 *************************************************************/

package com.terminal;
```

```
import se.ericsson.eto.norarc.javaframe.*;
import com.messages.*;

public class TerminalAgentCS extends CompositeState {

        static State Idle = new State("IdleTAS");
        static State Wait_Database = new State("Wait_DatabaseTAS");
        static State Result = new State("ResultTAS");
        static State Wait_Map = new State("Wait_MapTAS");
        static State Wait_Monitor = new State("Wait_MonitorTAS");
        static State Wait_Alarm = new State("Wait_AlarmTAS");

        /**
         * Constructor.
         * Declares the states of the LocationAgent.
         * @param sn
         */
        public TerminalAgentCS(String sn) {
                super(sn);
            Idle.enclosingState = this;
                Wait_Database.enclosingState = this;
                Result.enclosingState = this;
                Wait_Map.enclosingState = this;
                Wait_Monitor.enclosingState = this;
                Wait_Alarm.enclosingState = this;
        }

        /**
         * The first state is entered automatically
         */
    public void enterState(StateMachine fsm) {
        Idle.enterState(fsm);
    }

        /**
         * Method that executes transitions for the statemachine.
         * The current state and the input message defines what the next state
         * will be.
         * @param sig The current input signal
         * @param st The current state
         * @param curfsm The current final state machine
         */
    protected boolean execTrans(Message sig, State st, StateMachine curfsm){
                TerminalAgentSM ta = (TerminalAgentSM)curfsm;

        //****************STATE: IDLE*************************************
                if (st == Idle){
                        if(sig instanceof SearchForObjectsM){
                                SearchForObjectsM im=(SearchForObjectsM)sig;
                                performExit(ta);
                                System.out.println("**TermAgent(" + st.stateName + "): Sendt: " +
sig.messageName() + sig.messageContent());
                                output(sig,ta.TaToRout,ta);
                                Wait_Database.enterState(ta);
                        }
                        return true;
                }
```

```
        //**************STATE: WAIT_DATABASE*************************************

                else if (st == Wait_Database){
                        if(sig instanceof ResultListM){
                                ResultListM im=(ResultListM)sig;
                                performExit(ta);
                                System.out.println("**TermAgent(" + st.stateName + "): Sendt: " +
sig.messageName() + sig.messageContent());
                                output(sig,ta.TaToEnv,ta);
                                Result.enterState(ta);
                        }
                        return true;
                }

        //***************STATE: RESULT*************************************************

                else if (st == Result){
                        if(sig instanceof ShowMapM){
                                ShowMapM im=(ShowMapM)sig;
                        performExit(ta);
                                System.out.println("**TermAgent(" + st.stateName + "): Sendt: " +
sig.messageName() + sig.messageContent());
                        output(sig,ta.TaToRout,ta);
                        Wait_Map.enterState(ta);
                        }
                        else if(sig instanceof MonitorObjectsM){
                                MonitorObjectsM im=(MonitorObjectsM)sig;
                                performExit(ta);
                                System.out.println("**TermAgent(" + st.stateName + "): Sendt: " +
sig.messageName() + sig.messageContent());
                                output(sig,ta.TaToRout,ta);
                                Wait_Monitor.enterState(ta);
                        }
                        else if(sig instanceof StartAlarmM){
                                StartAlarmM im=(StartAlarmM)sig;
                                performExit(ta);
                                System.out.println("**TermAgent(" + st.stateName + "): Sendt: " +
sig.messageName() + sig.messageContent());
                                output(sig,ta.TaToRout,ta);
                                Wait_Alarm.enterState(ta);
                        }
                        return true;
                }

        //****************STATE: WAIT_MAP************************************************

                else if (st == Wait_Map){
                        if(sig instanceof MapWithObjectsM){
                                MapWithObjectsM im=(MapWithObjectsM)sig;
                        performExit(ta);
                                System.out.println("**TermAgent(" + st.stateName + "): Sendt: " +
sig.messageName() + sig.messageContent());
                        output(sig,ta.TaToEnv,ta);
                        Idle.enterState(ta);
                        }
                        return true;
                }
```

```
        //******************STATE: WAIT_MONITOR*****************************************

                else if (st == Wait_Monitor){
                        if(sig instanceof NewLocationM){
                                NewLocationM im=(NewLocationM)sig;
                                System.out.println("**TermAgent(" + st.stateName + "): Received:
" + sig.messageName() + sig.messageContent());
                                output(new ShowMapM(im.xcoord, im.ycoord),ta.TaToRout,ta);
                                sameState(ta);
                        }
                        else if(sig instanceof EndMonitorM){
                                EndMonitorM im=(EndMonitorM)sig;
                                performExit(ta);
                                System.out.println("**TermAgent(" + st.stateName + "): Sendt: " +
sig.messageName() + sig.messageContent());
                                output(sig,ta.TaToRout,ta);
                                Idle.enterState(ta);
                        }
                        else if(sig instanceof MapWithObjectsM){
                                MapWithObjectsM im=(MapWithObjectsM)sig;
                                System.out.println("**TermAgent(" + st.stateName + "): Sendt: " +
sig.messageName() + sig.messageContent());
                                output(sig,ta.TaToEnv,ta);
                                sameState(ta);
                        }
                        return true;
                }

        //********************STATE: WAIT_ALARM*************************************

                else if(st == Wait_Alarm){
                        if(sig instanceof AlarmM){
                                AlarmM im=(AlarmM)sig;
                                System.out.println("**TermAgent: Received: " + sig.messageName()
+ sig.messageContent());
                                output(new ShowMapM(im.xcoord, im.ycoord),ta.TaToRout,ta);
                                output(new AlarmToTerminalM(),ta.TaToEnv,ta);
                                sameState(ta);
                        }
                        else if(sig instanceof EndAlarmM){
                                EndAlarmM im=(EndAlarmM)sig;
                                performExit(ta);
                                System.out.println("**TermAgent: Sendt: " + sig.messageName() +
sig.messageContent());
                                output(sig,ta.TaToRout,ta);
                                Idle.enterState(ta);
                        }
                        else if(sig instanceof ShowMapM){
                                ShowMapM im=(ShowMapM)sig;
                                System.out.println("**TermAgent: Sendt: " + sig.messageName() +
sig.messageContent());
                                output(sig,ta.TaToRout,ta);
                                sameState(ta);
                        }
                        else if(sig instanceof MapWithObjectsM){
                                MapWithObjectsM im=(MapWithObjectsM)sig;
                                System.out.println("**TermAgent: Sendt: " + sig.messageName() +
sig.messageContent());
                                output(sig,ta.TaToEnv,ta);
```

```
                        sameState(ta);
                }
                return true;

        }

    //**********************END STATES************************************
        return true;
    }//execTrans

}//Class
```

G.3 package com.router

```
/*************************************************************
 * This is the State Machine for the RouterBlock.
 * It starts the execution.
 *
 * RouterBlockSM.java
 *
 * @author Solrun Pedersen
 *************************************************************/

package com.router;
import se.ericsson.eto.norarc.javaframe.*;

public class RouterBlockSM extends StateMachine {

    static CompositeState states = new RouterBlockCS("RouterBlock states");

    // Declare local references to output mediators
    protected Mediator RoutToTa;
    protected Mediator RoutToLa;
    protected Mediator RoutToDb;
    protected Mediator RoutToMap;

    /**
     * Constructor.
     * @param sched Sheduler for the system
     * @param RoutToTa Mediator from the Router to the TerminalAgent
     * @param RoutToLa Mediator from the Router to the LocationAgent
     * @param RoutToDb Mediator from the Router to the DatabaseProxy
     * @param RoutToMap Mediator from the Router to the MapProxy
     * @param RoutToSelf
     */
    public RouterBlockSM(Scheduler sched, Mediator RoutToTa, Mediator RoutToLa, Mediator
RoutToDb, Mediator RoutToMap, Mediator RoutToSelf) {
            super(sched);
            this.RoutToTa = RoutToTa;
            this.RoutToLa = RoutToLa;
            this.RoutToDb = RoutToDb;
            this.RoutToMap = RoutToMap;
            RoutToSelf.addAddress(this);     // connect the input Mediator
    }

    /**
     * Method to start the Transition of the state machine
```

```
         */
        protected void execStartTransition() {
                states.enterState(this);
        }
}
```

```
/*************************************************************
 * This is the Composite State for the Router.
 * It declares all the states, and which messages it can receive
 * and send in the different states.
 *
 * RouterBlockCS.java
 *
 * @author Solrun Pedersen
 *************************************************************/

package com.router;
import se.ericsson.eto.norarc.javaframe.*;
import com.messages.*;

public class RouterBlockCS extends CompositeState {

        static State idle = new State("IdleR");

        /**
         * Constructor.
         * Declares the states of the RouterBlock.
         * @param sn
         */
        public RouterBlockCS(String sn) {
                super(sn);
            idle.enclosingState = this;
        }

        /**
         * The first state is entered automatically
         */
    public void enterState(StateMachine fsm) {
        idle.enterState(fsm);
    }

        /**
         * Method that executes transitions for the statemachine.
         * The current state and the input message defines what the next state
         * will be.
         * @param sig The current input signal
         * @param st The current state
         * @param curfsm The current final state machine
         */
    protected boolean execTrans(Message sig, State st, StateMachine curfsm){
                RouterBlockSM rb = (RouterBlockSM)curfsm;

                if (st == idle ){
                        if(sig instanceof SearchForObjectsM){
                                System.out.println("**Router(" + st.stateName + "): Sendt: " +
sig.messageName() + sig.messageContent());
                 output(sig,rb.RoutToDb,rb);
```

```
                        sameState(rb);
                }

                else if(sig instanceof ResultListM){
                        System.out.println("**Router(" + st.stateName + "): Sendt: " +
sig.messageName() + sig.messageContent());
                        output(sig,rb.RoutToTa,rb);
                        sameState(rb);
                }
                else if(sig instanceof ShowMapM){
                        System.out.println("**Router(" + st.stateName + "): Sendt: " +
sig.messageName() + sig.messageContent());
                        output(sig,rb.RoutToMap,rb);
                        sameState(rb);
                }
                else if(sig instanceof MapWithObjectsM){
                        System.out.println("**Router(" + st.stateName + "): Sendt: " +
sig.messageName() + sig.messageContent());
                        output(sig,rb.RoutToTa,rb);
                        sameState(rb);
                }
                else if(sig instanceof MonitorObjectsM){
                        System.out.println("**Router(" + st.stateName + "): Sendt: " +
sig.messageName() + sig.messageContent());
                        output(sig,rb.RoutToLa,rb);
                        sameState(rb);
                }
                else if(sig instanceof NewLocationM){
                        System.out.println("**Router(" + st.stateName + "): Sendt: " +
sig.messageName() + sig.messageContent());
                        output(sig,rb.RoutToTa,rb);
                        sameState(rb);
                }
                else if(sig instanceof EndMonitorM){
                        System.out.println("**Router(" + st.stateName + "): Sendt: " +
sig.messageName() + sig.messageContent());
                        output(sig,rb.RoutToLa,rb);
                        sameState(rb);
                }
                else if(sig instanceof StartAlarmM){
                        System.out.println("**Router(" + st.stateName + "): Sendt: " +
sig.messageName() + sig.messageContent());
                        output(sig,rb.RoutToLa,rb);
                        sameState(rb);
                }
                else if(sig instanceof AlarmM){
                        System.out.println("**Router(" + st.stateName + "): Sendt: " +
sig.messageName() + sig.messageContent());
                        output(sig,rb.RoutToTa,rb);
                        sameState(rb);
                }
                else if(sig instanceof EndAlarmM){
                        System.out.println("**Router(" + st.stateName + "): Sendt: " +
sig.messageName() + sig.messageContent());
                        output(sig,rb.RoutToLa,rb);
                        sameState(rb);
                }
        }
                return true;
        }
```

```
}//class
```

G.4 package com.locationserver

```
/*************************************************************
 * This is the State Machine for the LocationAgent.
 * It starts the execution.
 *
 * LocationAgentSM.java
 *
 * @author Solrun Pedersen
 *************************************************************/

package com.locationserver;

import se.ericsson.eto.norarc.javaframe.*;

public class LocationAgentSM extends StateMachine {

    static CompositeState states = new LocationAgentCS("UserAgent states");
        protected Mediator LaToEnv;
    protected Mediator LaToRout;

        /**
         * Constructor.
         * @param sched Sheduler for the system
         * @param LaToEnv Mediator from the LocationAgent to the Location Server
         * @param LaToRout Mediator from the LocationAgent to the Router
         * @param LaToSelf
         */
    public LocationAgentSM(Scheduler sched, Mediator LaToEnv, Mediator LaToRout, Mediator
LaToSelf){
                super(sched);
                this.LaToEnv=LaToEnv;
                this.LaToRout=LaToRout;
                LaToSelf.addAddress(this);
    }

        /**
         * Method to start the Transition of the state machine
         */
    protected void execStartTransition(){
                states.enterState(this);
    }

}//class
```

```
/*************************************************************
 * This is the Composite State for the LocationAgent.
 * It declares all the states, and which messages it can receive
 * and send in the different states.
 *
 * LocationAgentCS.java
 *
```

```
 * @author Solrun Pedersen
 ************************************************************/

package com.locationserver;
import se.ericsson.eto.norarc.javaframe.*;
import com.messages.*;

public class LocationAgentCS extends CompositeState {

	static State Idle = new State("IdleTAS");
	static State Wait_Monitor = new State("Wait_MonitorTAS");
	static State Wait_Alarm = new State("Wait_AlarmTAS");


	/**
	 * Constructor.
	 * Declares the states of the LocationAgent.
	 * @param sn
	 */
	public LocationAgentCS(String sn) {
		super(sn);
		Idle.enclosingState = this;
		Wait_Monitor.enclosingState = this;
		Wait_Alarm.enclosingState = this;
	}

	/**
	 * The first state is entered automatically
	 */
	public void enterState(StateMachine fsm) {
		Idle.enterState(fsm);
	}

	/**
	 * Method that executes transitions for the statemachine.
	 * The current state and the input message defines what the next state
	 * will be.
	 * @param sig The current input signal
	 * @param st The current state
	 * @param curfsm The current final state machine
	 */
	protected boolean execTrans(Message sig, State st, StateMachine curfsm){

		LocationAgentSM la = (LocationAgentSM)curfsm;

	//***************STATE: IDLE*************************************
		if (st == Idle){
			if(sig instanceof MonitorObjectsM){
				MonitorObjectsM im=(MonitorObjectsM)sig;
				performExit(la);
				System.out.println("**LocationAgent(" + st.stateName + "): Sendt:
" + sig.messageName() + sig.messageContent());
				output(sig,la.LaToEnv,la);
				Wait_Monitor.enterState(la);
			}

			else if(sig instanceof StartAlarmM){
				StartAlarmM im=(StartAlarmM)sig;
				performExit(la);
```

```
                                System.out.println("**LocationAgent(" + st.stateName + "): Sendt:
" + sig.messageName() + sig.messageContent());
                                output(sig,la.LaToEnv,la);
                                Wait_Alarm.enterState(la);
                        }
                        return true;
                }

        //******************STATE: WAIT_MONITOR*****************************************

                else if (st == Wait_Monitor){
                        if(sig instanceof NewLocationM){
                                NewLocationM im=(NewLocationM)sig;
                                System.out.println("**LocationAgent(" + st.stateName + "): Sendt:
" + sig.messageName() + sig.messageContent());
                                output(sig,la.LaToRout,la);
                                sameState(la);
                        }

                        else if(sig instanceof EndMonitorM){
                                EndMonitorM im=(EndMonitorM)sig;
                                performExit(la);
                                System.out.println("**LocationAgent(" + st.stateName + "): Sendt:
" + sig.messageName() + sig.messageContent());
                                output(sig,la.LaToEnv,la);
                                Idle.enterState(la);
                        }
                        return true;
                }

        //********************STATE: WAIT_ALARM*************************************

                else if(st == Wait_Alarm){
                        if(sig instanceof AlarmM){
                                AlarmM im=(AlarmM)sig;
                                System.out.println("**LocationAgent(" + st.stateName + "): Sendt:
" + sig.messageName() + sig.messageContent());
                                output(sig,la.LaToRout,la);
                                sameState(la);
                        }

                        else if(sig instanceof EndAlarmM){
                                EndAlarmM im=(EndAlarmM)sig;
                                performExit(la);
                                System.out.println("**LocationAgent(" + st.stateName + "): Sendt:
" + sig.messageName() + sig.messageContent());
                                output(sig,la.LaToEnv,la);
                                Idle.enterState(la);
                        }
                        return true;

                }

        //**********************END STATES*****************************************
                return true;
        }//execTrans

}//Class
```

G.5 package com.messages

```
/*************************************************************
 * Message from the LocationAgent to the Terminal
 * telling an alarm has gone off for an object.
 *
 * AlarmM.java
 *
 * @author Solrun Pedersen
 *************************************************************/

package com.messages;
import se.ericsson.eto.norarc.javaframe.*;

public class AlarmM extends Message {

        public String xcoord;
        public String ycoord;

        /**
         * Constructor.
         * @param xcoord The x coordinate for the location of the alarm
         * @param ycoord The y coordinate for the location of the alarm
         */
        public AlarmM(String xcoord, String ycoord) {
        super();
                this.xcoord=xcoord;
                this.ycoord=ycoord;
        }

}
```

```
/*************************************************************
 * Message from the Router to the Terminal
 * telling an alarm has gone off for an object.
 *
 * AlarmToTerminalM.java
 *
 * @author Solrun Pedersen
 *************************************************************/

package com.messages;
import se.ericsson.eto.norarc.javaframe.*;

public class AlarmToTerminalM extends Message {

        public AlarmToTerminalM() {
                super();
        }

}
```

```
/************************************************************
 * Message from the Terminal to the LocationAgent
 * telling to stop receiving alarm for an object.
 *
 * EndAlarmM.java
 *
 * @author Solrun Pedersen
 ************************************************************/

package com.messages;
import se.ericsson.eto.norarc.javaframe.*;

public class EndAlarmM extends Message {

        public EndAlarmM() {
                super();
        }

}
```

```
/************************************************************
 * Message from the Terminal to the LocationAgent
 * telling to stop monitoring an object.
 *
 * EndMonitorM.java
 *
 * @author Solrun Pedersen
 ************************************************************/

package com.messages;
import se.ericsson.eto.norarc.javaframe.*;

public class EndMonitorM extends Message {

        public EndMonitorM() {
                super();
    }

}
```

```
/************************************************************
 * Message from the MapProxy to the Terminal
 * returning an image of a map.
 *
 * MapWithObjectsM.java
 *
 * @author Solrun Pedersen
 ************************************************************/

package com.messages;

import javax.swing.ImageIcon;
```

```
import se.ericsson.eto.norarc.javaframe.*;

public class MapWithObjectsM extends Message {

        public ImageIcon map;

        /**
         * Constructor.
         * @param map The ImageIcon containing the map
         */
    public MapWithObjectsM(ImageIcon map) {
                super();
            this.map=map;
    }

}
```

```
/************************************************************
 * Message from the Terminal to the LocationAgent
 * asking to start monitoring an object.
 *
 * MonitorObjectsM.java
 *
 * @author Solrun Pedersen
 ************************************************************/

package com.messages;
import se.ericsson.eto.norarc.javaframe.*;

public class MonitorObjectsM extends Message {

        public String unikID;

        /**
         * Constructor.
         * @param unikID The unique identification number of the object
         */
    public MonitorObjectsM(String unikID) {
                super();
            this.unikID=unikID;
    }

}
```

```
/************************************************************
 * Message from the LocationAgent to the Terminal
 * with a new location of a monitored object.
 *
 * NewLocationM.java
 *
 * @author Solrun Pedersen
 ************************************************************/

package com.messages;
```

```
import se.ericsson.eto.norarc.javaframe.*;

public class NewLocationM extends Message {

        public String xcoord;
        public String ycoord;

        /**
         * Constructor.
         * @param xcoord The x coordinate for the new location
         * @param ycoord The y coordinate for the new location
         */
   public NewLocationM(String xcoord, String ycoord) {
           super();
           this.xcoord=xcoord;
           this.ycoord=ycoord;
   }

}
```

```
/*************************************************************
 * Message from the DatabaseProxy to the Terminal
 * with a list of objects matching a search.
 *
 * ResultListM.java
 *
 * @author Solrun Pedersen
 *************************************************************/

package com.messages;
import java.util.ArrayList;

import se.ericsson.eto.norarc.javaframe.*;

public class ResultListM extends Message {

        public ArrayList result;

        /**
         * Constructor.
         * @param result An ArrayList containing the result of the search
         */
    public ResultListM(ArrayList result) {
               super();
        this.result=result;
    }

}
```

```
/*************************************************************
 * Message from the Terminal to the DatabaseProxy
 * asking to search for objects matching certain criterias.
 *
 * SearchForObjectsM.java
 *
```

```
 * @author Solrun Pedersen
 ************************************************************/

package com.messages;
import se.ericsson.eto.norarc.javaframe.*;

public class SearchForObjectsM extends Message {

        public String type;
        public String avd;
        public String status;
        public String navn;
        public String unikId;

        /**
         * Constructor.
         * @param type The type of the object
         * @param avdeling The ward that the object belongs to
         * @param status The current status of the object
         * @param navn Used when searching for persons. The name of the person
         * @param unikId The unique identification number of the object
         */
        public SearchForObjectsM(String type, String avd, String status, String navn, String
unikId) {
                super();
        this.type=type;
        this.avd=avd;
        this.status=status;
                this.navn=navn;
                this.unikId=unikId;
    }

}
```

```
/************************************************************
 * Message from the Terminal to the MapProxy
 * asking to get a map showing the location of an object.
 *
 * ShowMapM.java
 *
 * @author Solrun Pedersen
 ************************************************************/

package com.messages;
import se.ericsson.eto.norarc.javaframe.*;

public class ShowMapM extends Message {

        public String xcoord;
        public String ycoord;

        /**
         * Constructor.
         * @param xcoord The x coordinate for the location of the object
         * @param ycoord The y coordinate for the location of the object
         */
    public ShowMapM(String xcoord, String ycoord) {
```

```
                super();
        this.xcoord=xcoord;
        this.ycoord=ycoord;
    }

}
```

```
/*************************************************************
 * Message from the Terminal to the Location Agent
 * telling to start receiving alarms for an object.
 *
 * StartAlarmM.java
 *
 * @author Solrun Pedersen
 *************************************************************/

package com.messages;
import se.ericsson.eto.norarc.javaframe.*;

public class StartAlarmM extends Message {

        public String unikID;

        /**
         * Constructor.
         * @param unikID The unique identification number of the object
         */
    public StartAlarmM(String unikID) {
                super();
                this.unikID=unikID;
    }

}
```