Abstract

Location-aware services have become more important during the last decade due to the increasing mobility and connectivity of users and resources. Location-awareness is an important aspect of making an application context-aware. In supporting collaboration in a ubiquitous computing environment taking advantage of location information is an important feature.

The UbiCollab Platform is a platform that supports collaboration in a ubiquitous environment. This thesis presents an extension of the UbiCollab platform to make it location-aware. The work shows how the location service can be developed to handle storing and querying of location information.

Preface

This report was written for the Norwegian University of Science and Technology, Department of Computer and Information Science and Telenor Research and Development, Trondheim, Norway in the spring 2005 as a master thesis.

The report contains work on a location service for a ubiquitous collaboration platform. It is an extension to the previous work done on the UbiCollab platform by Christian Schwarz in the spring of 2004 and the work done by Carsten Andreas Heitmann and Børge Setså Jensen in the autumn of 2004. The report present an analysis, solution and demonstration of the location service.

I wish to thank my project supervisors, Professor Dr. Monica Divitini at Department of Computer and Information Science and Babak Amin Farshchian at Telenor for all valuable support and feedback during this work. I would also like to thank Carsten Andreas Heitmann, Hans Steien Rasmussen and Anders Magnus Braathen for good collaboration.

Trondheim, June 16, 2005

Børge Setså Jensen

Contents

List of Figures

Chapter 1

Introduction

UbiCollab is a platform for supporting group collaboration in a ubiquitous environment. The vision and idea of UbiCollab is described in an article by Divitini *et al.* [DFS04]. Work on the theoretical aspects of the platform and a prototype have been done by Schwarz [Sch04]. A review of this work can be found in appendix A.

Ubiquitous computing is a relatively new research area. It was introduced by Mark Weiser and his colleagues at the Ubiquitous Computing project at Xerox PARC. Mark Weiser described it in the following way [Wei93]:

> Ubiquitous computing is the method of enhancing computer use by making many computers available throughout the physical environment, but making them effectively invisible to the user.

The field is constantly undergoing changes as new techniques and technology for ubiquitous computing are developed. The potential of location-aware services has increased as the users and resources become more and more mobile and connected. This master thesis will focus on location-awareness and how it can support collaboration in a ubiquitous computing environment.

1.1 Location awareness

Location has become an important aspect in ubiquitous collaboration because of the increased mobility that are available with todays technology. In a ubiquitous collaborative setting it can be important for the participating users to be aware of the location of other users. This can enable them to see the availability for cooperation in a given situation. In a ubiquitous computing environment it is essential that the user gets access to resources in the surrounding environment, without prior knowledge of the resources and their configuration. Location-awareness can enable the system to display and use resources that are available in the environment without need for specific configuration.

The research on use of location information in computer systems have mostly focused on single user systems and neglected cooperative settings. However recent research in CSCW (Computer Supported Cooperative Work) has taken up issues with using location information in a cooperative setting. The results of this research shows that location information is an important aspect of the information needs of the users. CSCW literature has often focused on the distinction between space and place [HD96]. Place is a much encompassing term than space. In CSCW literature space is generally meant to indicate the physical surroundings of the user whilst place also indicates social constructs to which people associate meaning, norms of behavior and the like.

As with the distinction between space and place in CSCW research I distinguish between location and position in this master thesis. Location in respect to this thesis is defined as the physical location an object has from the users point of view. Position can be defined as the physical position of an object, often presented in coordinates. Location contains more meta-information than position and can change depending on who is using it. For instance a room as a location can be viewed as a lunchroom by one entity but as a meeting room by another, but the position of the room is constant. Physical location can be represented in several different forms. Several standards and practises exist for representing location and position, I will discuss these in a later chapter.

Location services might be used to support a wide range of situations. It is important to understand location information in a wider context to support pervasive cooperation. Location services might be used to inform the user, it might be used to tailor interaction with another user or it might be used directly by the system to improve the user experience. One of the aims of this thesis is to show how UbiCollab can be extended to provide basic location services that can be used by applications running on the platform. Another important aspect when building location services for a cooperative setting is that they must be cooperation-aware. That is the location service should be aware of the collaboration instance it is providing the service to and tailor the information to the given collaboration instance's preferences.

1.2 Research focus and project goals

The aim of this master thesis is to create a foundation for handling location-awareness in the UbiCollab platform. The main focus of the research is on location-awareness and location models in a collaborative environment. Research into former systems, their architecture and different technical solutions must be undertaken to give a general understanding of the field. This will also give an understanding of how to create a location-aware application and enable me to design a location service that can handle location information in addition to position information. At the end of the thesis a functioning prototype showing some of the location-aware features discussed will be developed. A goal is that the architecture, location- and position-representation should be general and flexible enough to use different kinds of systems for position pinpointing.

The development of location-awareness should not compromise any of the previously developed functionality of UbiCollab. Since this master thesis is written in collaboration with two other groups, one working on the UbiCollab platform with privacy issues and the other working on shared displays using UbiCollab, it is important that the work is coordinated with their contribution.

1.3 Research method

The research method for this master thesis is scenario-based and prototype driven. The scenario used in this thesis is described in the next chapter. The scenario is used throughout the thesis as an example of how the UbiCollab platform is used and to highlight what the location service should be able to provide of functionality. The prototype will be described in a later chapter. At the start of the thesis time will be spent on researching the field of location-awareness to give a general understanding of the problem. Then as the level of understanding grows, more specific issues will be researched to help develop the proper solutions.

1.4 Relation to previous UbiCollab work

This master thesis is directly based on the previous work done on the UbiCollab Platform by Schwarz, Gonçalves and Bakkevold [Sch04, Gon04, Bak04] in their master thesis. The work is also based on the work done on the UbiCollab platform by Heitmann and Jensen [HJ04] in 2004. In this work the focus was on the position service and location service of the UbiCollab platform and a teoretical solution was presented as well as a prototype. The aim is to build upon the previous work by further developing the existing platform and only changing the architectural aspects when it is necessary to accomplish the goals. This should be done without removing any functionality already provided.

1.5 Report outline

The rest of this report is organised in the following chapters:

2 Problem elaboration: Presents an elaboration of the problem for this master thesis based on a scenario. A discussion of the scenario is used to highlight important aspects of the problem. In the end a list of research questions and expected results is presented.

3 Related work: Gives an overview over the field of location-awareness in a mobile computing environment and presents related work on representation and storing of location information. It also presents research on database solutions for handling spatial information.

4 Analyses and design: Gives and overview of the location services relation to other services. Presents the datamodel and API design for the location service.

5 Prototype: Describes how the location service is realised within the UbiCollab prototype.

6 Evaluation: Evaluates the theoretical and practical work. Gives a critical view on how the work process has been throughout the thesis.

7 Conclusion: Presents the conclusions drawn from the project, a summary of the contributions and present future work.

Chapter 2

Problem elaboration

This chapter aims to give a deeper understanding of the problem. First an introduction to the UbiCollab architecture and where the location service fits in this architecture is presented to give an understanding of the service and its purpose. Then the scenario that is used as a guide for this project is provided. Some points from the scenario of special interest and a description of the issues they represent are further described. Then a description of how the work done on this master thesis fit into the previous work and the work being done by the other groups are presented. To round of the problem elaboration the major research questions and expected results are presented in the final section.

2.1 UbiCollab architecture

This master thesis will focus on the location service in the UbiCollab system. Most of the work done will focus on the API the location service provide to the collaboration service and how to use and integrate location in the collaboration service. See Figure 2.1 for a conceptual overview of the different services that are present in UbiCollab. The services surrounded by dotted lines is where the focus of this thesis lies. The location service aim to provide the other services with location information on users, devices and other resources. It will communicate with the collaboration service and also the position service. The position service is a freestanding service that is meant to be implemented on pinpointing devices. The position module in the location service will then be able to communicate in a uniform fashion to all pinpointing devices through their position service. It will then provide position information on devices and resources that are inside the area of the pinpointing device's sensors. For a more thorough description of the services in the UbiCollab platform see section 4.2 of "Ubicollab: Improving collaboration with location services" [DFJ05].

In Figure 2.2 a more detailed view of the services in focus and their relationship is

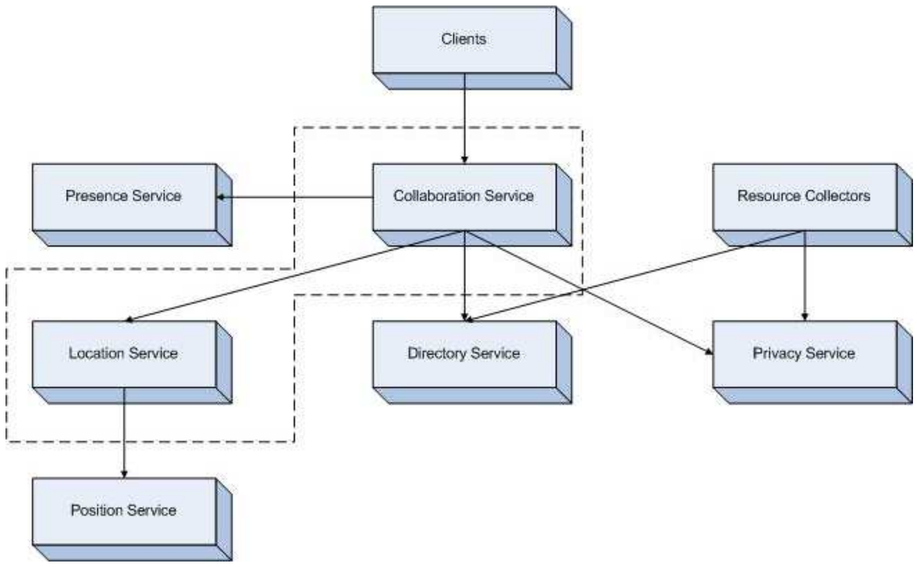


Figure 2.1: Architecture overview

presented. To integrate the location service in the UbiCollab system a functional and clear API for the location service must be defined so that the other services have a clear interface through which they can access the services provided by the location service. This thesis will focus on creating a complete API to enable full use of the location services from the other services such as the Collaboration service. Another aspect that will be looked into is how location services can be used in the Collaboration service to support cooperation.

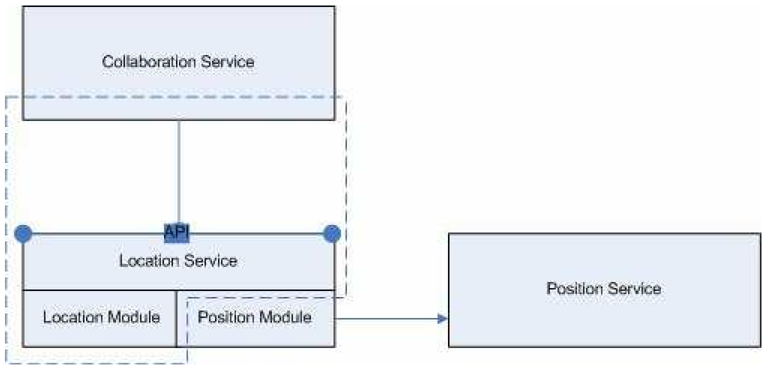


Figure 2.2: Location Service

How the location service handles each request from other services will not be the most important aspect of this thesis, but some work most be done on the internal workings on the service in order to produce a functional prototype.

2.2 Scenario analysis

The scenario that is used throughout this project is presented in the following section. I have tried to create a scenario that presents some of the interesting ways location information can be used to promote collaboration in UbiCollab. The scenario is used as a guide to functionality that needs to be implemented in the prototype and what has to be simplified or left out. The scenario is also provided to give an overview of how UbiCollab can use location information to enhance cooperation.

> Christian, Amanda and John are all working in a telecommunication company. They are currently working together on the same project, but due to nature of their work they spend much time working in separate geographic locations. To help minize the effect of the distance between them they use the UbiCollab platform as a tool to support collaboration.
>
> Christian is the project manager and he wishes to have a meeting with the rest of the project group. He is currently located in the headquarters in oslo. He logs on to UbiCollab with his PDA client and creates a meeting. The screen asks for time, place and people. He schedules a meeting with Amanda, John and the rest of the project group at 12 o'clock the next day in his office. He prepares some slides for the meeting and adds all relevant files to the meeting in UbiCollab so that the others can have a look at them.

Here we can see how the platform must support setting up a meeting for a collaboration instance. This is already implemented in the state the prototype is in at the start of this master thesis [HJ04].

> Amanda and John are as often is the case away from headquarters on business. They both receive an email announcing the meeting, they log on to UbiCollab and review the files associated with the meeting.
>
> The next day the project group gathers in Christians office for the scheduled meeting. When they enter the meeting room, the UbiCollab system registers their location and set the availability of the participants according to the setting of their meeting profile. On the UbiCollab screen Christian can see information on who is present at the meeting, where they are located in the world and how they are connected to the UbiCollab platform. With a few mouse-clicks he can also check which devices and resources each participant has available in his surroundings.

Here we see how the platform register location information and then display the information for users with access to see. Christian get to see not only who is present and where they are located, but also information about their context and surroundings.

One of the challenges with this is to be able to gather the proper information and displaying it in a suitable way. The system must find the position of devices, resources and people. Then it must convert the positions into locations based on the context of the entities. How to display the information in a simple and understandable way is also an important issue.

> Because of the short notice for the meeting Amanda and John could not reschedule their travel arrangements for this day. On his UbiCollab meeting screen Christian can see that John is at the airport and is connected through a wireless on his laptop. Amanda on the other hand has had a small delay and is on her way to the headquarters in a taxi. Amanda is connected through her cellphone and has limited functionality available to her.
>
> Because Amanda is connected Christian can see her status and also get some location information. He can see that she is in a car moving towards headquarters and he sends her a message proposing that they start the meeting discussing issues that do not require her presence.

This part of the scenario present an interesting aspect with mobile users and limited resources available. Amanda is only logged in through her cellphone but it may still be able to provide the system with some information that can enhance cooperation. Giving Christian the ability to see Amandas location enables him to plan the meeting accordingly and start with issues where her presence is unnecessary.

> As the meeting progress John can follow the presentation slides as they are shifted on a shared display he has set up on his laptop. He can also control the slideshow if he needs to go to a specific slide to discuss something. Amanda can also follow the progress of the slideshow from her seat in the taxi. She has printed the slides on paper and the shared display she has set up on her phone continuously updates her on which slide the others are looking at.
>
> Christian has some information he wants John to have a look at during his flight. He can see on the UbiCollab screen that John has a publicly available printer in his surroundings and tells John he will send the information to the printer. John is not sure where this printer is located, but he pulls up the printer information on the UbiCollab screen and see that the printer has provided location

> information. Using the location service in UbiCollab he gets a map showing his own location and the location of the printer.

The platform must enable the users to get the most out of the equipment they have available in their surroundings. We can see how Amanda is able to follow the slideshow even though she is only connected with her phone. Being able to adapt the information being displayed to the equipment it is being displayed on is an important aspect but will not be a main focus of this thesis.

The other interesting issue in this part of the scenario is how UbiCollab helps John find an available printer. We see that the system enables Christian to send information to a printer in Johns surroundings. To enable John to be able to locate the printer with help of the location service there are several issues that must be looked into. First of all how will the location service get location information on the printer. In the scenario we see that the printer has provided some location information with its service description. When the devices do not provide location information one solution would be using the position service to produce location information on a device. Another issue that is interesting is how to convert the location information available on the printer and on Johns position into a suitable representation that can be used by John to navigate himself to the printer.

> Pete is working on a different project together with Christian. He logs on to UbiCollab to see if he can have a few words with Christian. He can see that Christian is in his office but is currently sitting in a meeting on another project. He sends a message to Christian telling him he wants to discuss some issues when Christian finds the time for it. The message will be delivered to Christian's screen as soon as the meeting has ended.

This last part of the scenario presents some interesting issues with regards to what information should be presented to other users and also how location and context information can be used to simplify user interaction. Pete who is not directly attached to the project working on at the moment will not get the full location information but rather a more general description of what he is doing. Pete also gets the option of sending a message that will be delivered as soon as Christian's context and location imply that the meeting is over.

2.3 Relation to previous and ongoing work on UbiCollab

As mentioned in the introduction, location services can be used to support a range of different situations and needs. The development of wireless capabilities together with the possibility of finding the position of wireless devices has opened up a whole new set of possibilities for location services.

In the previous work on UbiCollab a shared display application has been developed. Carsten Heitmann is continuing the work on this application and several interesting issues can be looked into with regards to presenting location information on a shared display.

One interesting aspect would be providing the shared display with location information to display a map over the people in a collaboration instance and their position. The map of people could be displayed as a standard geographic map or as a textual map with description of peoples location. To be able to provide this information the location service would have to keep a database that contains information of the locations users might be in. The location service must be able to convert a position in coordinates into a location. Given a coordinate the service must search to the database to see if it fits into one of the known locations.

The other group that is working on the UbiCollab platform is working on privacy issues. Their focus is on creating a privacy extension that enables the platform to adapt what information it provide to users based on privacy settings on devices, resources and user profiles. The work on this thesis will not be directly influenced by the privacy extension but in the prototype chapter issues that needs to be resolved for the prototype to be compatible with the work will be presented.

Another aspect of this master thesis will be how to integrate the location service with the other services of the UbiCollab platform. The UbiCollab platform is arranged as freestanding UPNP-enabled services so that services can be started and stopped independently. This makes the integration of the service easier since it is not directly dependent on the other services. However it is important that the API for the location services is clear and complete so that the other services can easily use it without needing to consider whats going on inside the service.

With the implementation of the location service there are several important issues that must be looked at. One of the issues is how to store the location information as well as making it available in a sensible way. There are two main solutions for this issue: using an external database tool that is capable of handling spatial queries or building the needed functionality on top of a simple database structure. In the next chapter on related work some available database solutions will be presented and evaluated. Another issue that will be discussed is how to create logical maps so that position data can be converted into location data.

2.4 Research questions and expected results

One of the main goals of this thesis is to develop a solution to the integration of the location service into the UbiCollab platform. To do this a clear interface towards the other services must be developed and this should hide all the inner workings of the location service, so that it is easy to use for the other services. The goal is to develop a teoretical solution and then implement it or a part of it in a prototype that will demonstrate the use of the location service. Another goal is to show how the position information gathered by the position module can be converted into location information dependent on the context of the user. To be able to do all this some areas require further research:

1. How should location information be stored and managed?
2. Which solutions are available for storing and querying location information?
3. What kind of architecture for position and location data should be used?
4. How to create logical maps and convert position data into location data?
5. How can the collaboration service utilize the location information provided by the location service?

The end result of this master thesis will be the teoretical results that are presented, in addition to a demonstrator which will show how the location service can be integrated into the UbiCollab prototype. The main goal of the thesis is to show how location information can be provided to different services through a specific API to the location service and how the other services can make use of this information.

Chapter 3

Related work

In this chapter I look at the concept of location and position representation and different architectures for handling location. The chapter starts of with a brief introduction of ubiquitous and context aware computing. Then some research into representation of location and position is presented. I will also look at the previous work on the UbiCollab platform and particularly with regards to support for location- and context-awareness. The chapter also present some existing database solutions that support storage and management of location data. The purpose of this chapter is to present research into how position information can be converted into location information and also how existing database solutions can be utilised to achieve this.

3.1 Introduction

Computers are becoming an ever greater part of our lives. They become smaller and more ubiquitous every time a new model is produced. The computer is becoming a part of most peoples everyday life, and this has had and will have enormous impact on the development of systems and applications in computing. The user can no longer be expected to have knowledge of specific computer systems. This has created a whole new set of requirements for user interfaces and interaction with the user. Context-aware applications can help lower the threshold for using an application. This because a context-aware application can offer an user interface that is adapted for the given situation. A context-aware application can be able to automatically configure itself and offer fewer and more relevant options for the user. This should not limit the users ability to interact with the application, only make it easier for the user to choose the correct options.

The last decade context-awareness and ubiquitous computing have become more and more important within research in computers. One of the reasons is that the advances within hardware have created more support for context-awareness, location-positioning and ubiquitous computing. Increased availability of wireless networks has made it possible to be online almost everywhere and enables computers to get and send information to increase context-awareness (e.g. find position by positioning over a WLAN-network).

3.1.1 How position and location relate to context and context-awareness

Position is in many settings the most important piece of data needed to make an application context-aware. If the application has knowledge of where the computer it is running on is located, it can in many instances be able to deduce what settings the user would prefer.

Often the users context can be deduced from several sensors and several data stores. However without knowing the position the process of deducing the context can be very hard. Position is usually tightly connected with context, that is what the user does and how he does it depends on where the user is. People organise their lives so that they have specific places for specific activities. When a person is at the office he works (preferably), when he is at the movies he is unavailable, when in the kitchen he cooks etc. Of course this is a bit of a simplification since a person can be in different contexts in the same location, but position is an important factor in deducing the context.

Position is so much more important in a mobile computing environment than in a stationary one because it is constantly changing. Before the computing devices became mobile the position did not really matter much because it was constant. Since the development of laptops, PDAs and enhanced mobile phones the position for where computing is done has become a dynamic variable.

3.2 Representing location and position

Christein and Schultess [CS02] define location in the following way:

> A location is a virtual or real place where someone or something is located at or is *present*. Examples of location in the virtual world are documents or host. (...) Examples of locations in the real world are postal addresses or GPS coordinates.

In my definition of location and position I say that location is the physical location an object has from the users points of view. The position is the physical position of an object often represented as coordinates. Unlike Christein and

Schultess I separate between location and position and give location a deeper meaning when I associate it with the context i.e. user viewpoint.

There are two main schemes for representing position in a computer system. The first one is the coordinate model, where position is represented through coordinates that map to the real world (e.g. 3D GPS coordinates). The other model is the hierarchical model [JS02]. This model is easier understood by humans because it decomposes the physical environment into different levels of precision and normally features a self-descriptive position representation (e.g. Position described with city, building, floor and room). The hierarchical model will be ambiguous with respect to position because users will have different perception of the meaning of elements in the hierarchy (Room 4 can be lunchroom 110 for one user and be meeting room 4 for another user.)

The two models have different advantages. With the coordinate model it is easy to calculate the exact distance between two entities, this is not so easy with the hierarchical model. However the hierarchical model can have support for implicit representation of spatial relationships such as closeness, containment, etc. while this is very hard in the coordinate model.

When designing a location service for the UbiCollab platform it is important to represent the position of entities in a general way, so that most pinpointing sources can be used by the platform. In the following sections some articles that have looked at representation of location and location models is presented.

3.2.1 On location models for ubiquitous computing

[BD04] is an article on different location models and their pros and cons. The article describes several location models that are interesting for this thesis and a summary of the article and its findings follows.

Introduction

Location information is presented in different formats, such as geometric coordinates and symbolic coordinates. There are three types of queries whitch should be supported by the location models; position, nearest neighbor and range. The suitability of a location model for distinct queries depnds on its internal organization.

Basic properties of coordinates

The article defines coordinates in the following definition [BD04]:

> A *coordinate x* is an identifier which specifies the position of an object with respect to a a given *coordinate system*. A coordinate system is a set X of coordinates.

Examples include:

- Geographic coordinates in the WGS84 used by GPS: triples containing longitude, latitude and elevation.
- The active bat system: high-resolution indoor positioning system, three dimensional coordinates with respect to local Cartesian reference system.
- The active badge system: symbolic identifiers for locations via IR.

From these examples two basic classes of coordinates can be identified: geometric and symbolic coordinates.

Geometric coordinates: define positions in the form of coordinate tuples relative to a reference coordinate system. Geometric coordinates can be divided into global and local geometric systems. The World Geodetic System 1984 (WGS84) is a global reference system while the Cartesian coordinate system of the active bat system can only define local coordinates. Geometric coordinates allows for calculation of distances between locations and deduction of topological relations such as spatial containment. Hence geometric coordinates already allow simple spatial reasoning.

Symbolic coordinates: define positions in the form of abstract symbols such as sensor identifiers, building and room names, etc. Symbolic coordinates do not provide any direct means for calculating distances or other spatial properties. So the symbolic location models have to provide additional information on the symbolic coordinates to define spatial properties.

Requirements for location models

The article defines three different types of queries that users/application might pose to a location model:

Position queries: All location models contain information on objets position, but they differ in the way it is represented. The definition of position requiers some form of coordinates. A general location model has to support different coordinate reference sysmtes, global and local ones.

Nearest neighbor queries: A search for the n objects closest to a certain position, i.e. searching for the closest printer. To support this the location model must contain not only information on the positions of the objects, but also define a distance function. The distance function makes it possible to compare the distance between two objects to the distance between other objects. In a geometric coordinate system this distance can be calculated by the physical distance between the objects. In a symbolic coordinate model the distances between coordinates have to be explicitly defined.

Range queries: A range query returns all objects within a certain geographic area. Such a query might be finding all the users within a buildings area. To answer such a query object positions must be known and also the topological relation “contains” has to be modeled. For geometric coordinates this information can be derived from the known geometry, but for symbolic coordinates it has to be defined explicitly. A symbolic model could define a room as being on (“within”) a floor which in turn is part of (“within”) a building.

From the cases presented above the article arrives at the following requirements for location models. A location model should provide:

- Object positions: Positions of objects have to be modeled in the form of coordinates. Supported coordinate and reference systems are: geometric and symbolic coordinates and multiple local and global coordinate reference systems.

- Distance function: Distances between spatial objects have to be modeled. This can also be the “size” of a location e.g. the length of a road segment represent the distance one has to travel when crossing this location in order to reach another location.

- Topological relations:
  Spatial containment in order to allow range queries.
  Spatially connected to for navigation services.

- Orientation: In addition to the positions of mobile objects, the orientation in the horizontal and/or vertical dimensions can be supported.

Geometric location models

The geometric models describe locations through geometric figures. If several coordinate systems are used, the position and orientation of the different local and global systems have to be defined in order to translate coordinates from one system to another. All geometric models support the information to derive the topological relation “contained in”. However the “connected to” relation has to be modeled explicitly since there is no direct support in a geometric system. This information can be used to improve the distance function, for instance incorporating the actual distance a user has to travel, not only the direct distance between two objects.

Symbolic location models

There are three main models in presented in this section; the set-based model, the hierarchical model and the graph-based model.

Set-based model:
Based on a set L of symbolic coordinates where locations are defined by subsets of L. For instance a building with several floors and rooms could be modeled by letting the set L represent all the room numbers in the building. The second floor is then modeled as the subset containing all the rooms on the second floor. Arbitrary locations may also be defined by sets that contain one ore several rooms. The containment relation can be determined in a set-based model by looking at the intersection of two sets. For the sets L_1 and L_2 if $L_1 \cap L_2 \neq \emptyset$ then L_1 and L_2 overlap and if $L_1 \cap L_2 = L_1$ then L_2 contains L_1. A set based model can only model qualitative distance, that is it can model a neighbour relation and through this deduce that the distance between A and B is bigger than the distance between A and C. However it can not deduce the absolute quantitative distance.

Hierarchical model:
Like the set based model a hierarchical model consists of a set of locations L. The special thing about the hierarchical model is that the locations are ordered according to the spatial containment relation. That is a location L_1 is modelled as the ancestor of another location L_2 if L_1 contains L_2. Because of this the hierarchical models support range queries naturally. The hierarchical models can be seen as a special case of the set based models and can only support qualitative distance queries.

Graph-based model:
In the graph-based approach the symbolic coordinates define the vertices V of a graph G = (V,E). The edges of the graph represent direct connection between locations. Edges and/or vertices can be weighted to model distance. The graph-based model explicitly models the topological relation “connected to” and is therefore well suited to handle nearest neighbor queries. The graph-based model however has no direct support for modeling ranges such as buildings or floors. Range queries can be answered in the case of finding all objects within a radius of a reference object, but not finding all locations contained within a bigger location (range).

It is possible to combine the set-based and graph-based model to better support queries for range and distance. Such a combination would combine the benefits of both models and one would also be able to generate different views based on level of detail see Figure 3.1.

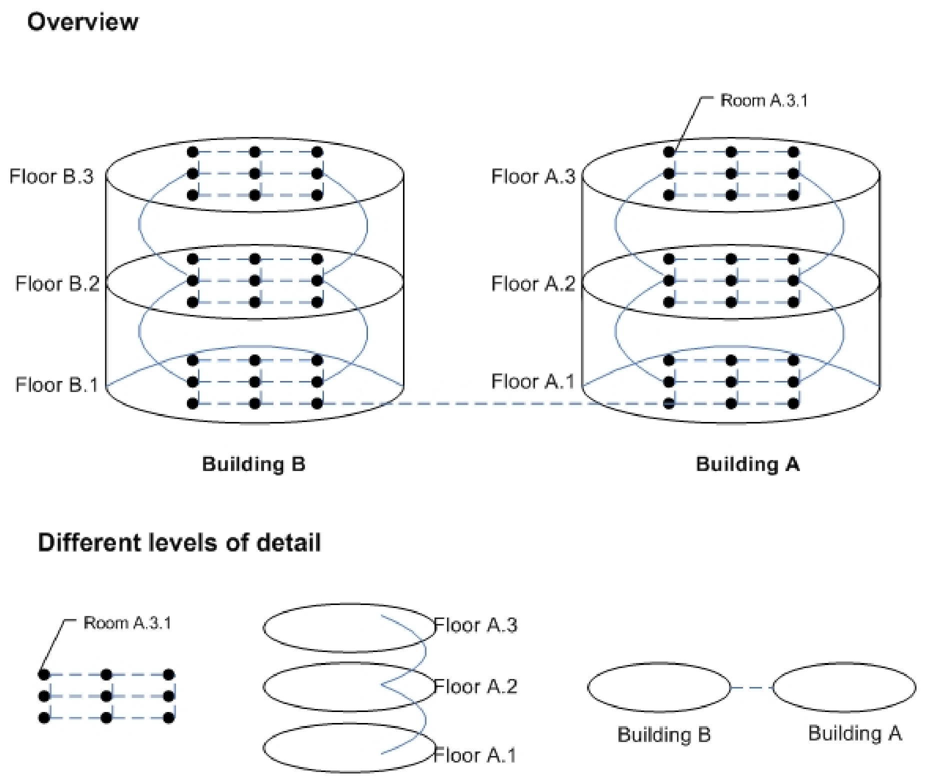


Figure 3.1: Set/graph-based model

Hybrid location models

The article presents two types of hybrid location models, the subspaces model and the partial subspaces model.

Subspaces This model uses a symbolic model like the combined symbolic model presented in the previous section and extends this with geometric model properties. That is for each location the geometric extent of the location is also stored in the location model. The geometric extent can be defined in accordance to a global reference system such as WGS84 or it can be defined in a local reference system where the coordinates have only local validity. Subspaces are formed by embedding coordinate systems into other coordinate systems by defining the position and orientation of embedded systems, a mode detailed description of this is found in [JS02] which is summarized in the next section.

Partial subspaces The second type of hybrid model is similar to the subspaces approach, but it does not assume that the geometric extent for every location is modeled. For instance the geometric extent of buildings might be included in the models but within the buildings only the symbolic models are used. This gives the benefit of enabling range queries for larger geometrically defined ranges (e.g. polygon on city plan) and it is also possible to do approximate calculations on user position based on the geometric extent of buildings and the symbolic mapping of rooms.

Discussion

In this article by Becker and Dürr a very clear description of the different location models and their strenghts and weaknesses are presented. From the article it is easy to see that no single model can solve all the possible requirements to a location model but rather the developer must select the model best suited to the application/usage. The hybrid location models presented in the end of the article are very interesting because they have the highest ability of solving the queries defined in the article. However the modeling effort for using the models increase with the higher complexity of the hybrid models.

3.2.2 A hybrid location model with a computable location identifier for ubiquitous computing

[JS02] describes a model for representing the location or position of an entity. It describes a position identifier that is a hybrid between the two different models for representing position (hierarchical and coordinate). This model would fit into the subspaces category of models defined in the previous section.

Hybrid model: Starting point hierarchical location model. View the world as hierarchy of spaces and each level further refines and subdivides the spaces of the previous level. Bring in the coordinate location model by allowing each space in the hierarchy to define a coordinate system that can be used to define points or areas within that space.

Use three types of location:

- *Space:* Location is a physical space entity, e.g. "room 3115 of 3rd floor of Wean Hall at CMU".
- *Area:* Location is a space not physically demarcated, but virtually defined by applications, e.g. "the area covered by a particular wireless access point".
- *Point:* Location is a position of mobile user or object.

Use formatted string representation complying with generic syntax of a Universal Resource Identifier to describe location:

- *Space identifier:* "ali://cmu/wean-hall/floor3/3100-corridor/3115".
- *Area identifier:* "ali://cmu/wean-hall/floor3{(1,0), (-1.5,0.5), (0,3), (2,3.5), (3,1.5)-(1,5)}".

- *Point identifier:* "ali://cmu/wean-hall/floor3/3700-corridor/3718(10,4,1)".

Each subspace has the following geometric attributes to integrate the coordinate model with the hierarchical model:

- *Shape:* Indicates the geometric shape of space.
- *Extension:* Combined with Shape attribute, specifies the volume/area covered by space.
- *Origin:* The origin point for the space coordinate system of the current space relative to the parent's coordinate system.
- *Rotation:* Matrix which specifies the directions of the three axes of the space coordinate system of the current space relative to the parent's coordinate system.

Discussion

Combining the location and position representation in one model can ease the way to resolving the position and location. It can also be used for a system that should work both in indoor and outdoor settings and be able to take advantage of several different sources for pinpointing. It would be able to resolve position and location information suitable for the operation in focus depending of what kind of position and location information wanted. However the representation is quite advanced and will be hard to implement in the UbiCollab system. The idea of different sets of location such as space, area and point will be considered when building logical maps.

3.2.3 Eploiting Space and Location as a Design Framework for Interactive Mobile Systems

[DRD$^+$00] present a design framework for location awareness in mobile systems. The article describes several interesting aspects with regards to location and its relation to context.

The article has a main focus on location and its implications when developing a interactive mobile system. To understand the role of location in relation to context it is necessary to look at how different forms of context influence interaction with mobile systems.

Different forms of context

Context can be divided into four groups:

- *Infrastructure context:* In mobile systems the nature of the infrastructure is likely to change as the application is used. User interfaces to mobile applications must be designed to cope with a level of uncertainty that comes from working with wireless communication and distributed information.
- *System context:* The extent to which a device is aware of other devices in its vicinity and also the extent to which an application is aware of other applications.
- *Domain context:* How the relationship between the users and mobile devices support the nature of the work being done. Another aspect is the level of trust and mutual awareness between participants in collaborative interactions. Especially if devices are used to identify users and potentially provide location and context information for others.
- *Physical context:* How mobile systems are aware of or embedded into their physical surroundings. If they are have sensory information about the surroundings this can be used to provide information or adapt the services presented to the user.

Location is very important in understanding context. For instance location is a useful indexing device from which to infer the overall context influencing the mobile application. To be able to find what devices are near this device (system context) we need to know the location of the device. To be able to measure the environment (physical context) the device must be physically located in space.

Location and Space

Any notion of location puts the device within some form of space. This space can contain other devices, users and applications with which the device can interact. The device interacts with the space in several ways:

- It has a location in the space.
- It has an effect on the space (interacting with devices and users within it).
- It is subject to influencing events from the space.

In a static system where the devices are fixed at a location the nature of interaction between devices would be one of configuration. In a mobile system however there is a much more dynamic relationship and the model for spatially situated interaction must take the following into consideration:

- Location in space (of the device and other bodies).
- Mobility through space (of these).

- The kinds of bodies populating the space (which the device may interact with).
- The awareness (of the device) of these other bodies.

Real and Virtual Worlds and Spaces

Computers and mobile devices existence and presence can be thought of in terms of many spaces. They can be considered as simultaneously inhabiting a real world and some forms of virtual worlds. Virtual worlds exist in almost all computer systems. It has grown from the use of spatial metaphors and techniques to represent information and actions. Development of cooperative systems in CSCW has relied much on concepts drawn from spatial arrangements. The latest research point in the direction of combining the real and the virtual. Examples include wearable computing and augmented reality. The fact that devices have both physical and virtual locations is something that must be considered when developing models of space and location.

A Taxonomy of Location

Any simpel mobile device will have a physical location in space. For some devices the exact Cartesian position in 2D or 3D space is important for defining a sense of absolute physical location. For others a more topological idea of space is sufficient in understanding position, and location is considered not in an absolute sense but in relation to other objects or sensors. This separation of location into cartesian spaces and topological spaces is also usable for virtual locations. For instance a location in a hypertext system might be defined using topological space and a location in virtual reality could be defined using a cartesian space. These categories are not mutually exclusive as a device may have both a precise longitude and latitude and have existence in one or more virtual spaces as well as the physical space. Many of the most interesting interaction possibilities occur when the different ideas of location are linked together. For instance moving a display up and down in the physical space could be used to change what it is displaying in its window.

3.3 Review of UbiCollab platform

The domain of the UbiCollab platform is the intersection of ubiquitous computing and collaboration support systems. In the previous work done by Schwarz, Bakkevold and Gonçalves a prototype of the platform, shared display and a PDA-client was developed. The master thesis by Schwarz [Sch04] has focus on specifying a complete platform for collaboration in a ubiquitous computing environment and this review will focus mostly on his work with regards to the location service.

To represent the abstract collaboration concept Schwarz uses a *collaboration instance*. This is an entity which is meant to capture a real world activity, or context, of collaboration between people and resources they use. A person entity represent a human participant while a resource can be a physical device or electronic information.

For a more complete introduction and review of the previous work on the UbiCollab platform see appendix A.

3.3.1 Location service

In the previous work on UbiCollab before autumn 2004 little focus was given to the location service. Schwarz has in his master thesis one section describing the service in the chapter on the theoretical foundation of the platform. He starts this section with an introduction [Sch04]:

> A location service is an important service when supporting ubiquitous collaboration, as it makes it easier for the users to find people to collaborate with or devices to use for collaboration.
>
> The UbiCollab location service builds on other more specific location services such as GPS, GSM and WLAN positioning services, and present the location in a useful way to clients of the service.

He goes on to describe some of the challenges of such a location service. One of the challenges he describes is that the lower level position devices can return the position in several different representations. Often these representations can be incompatible such as GPS coordinate, coordinate relative to some position, hierarchical position etc. This can make it harder to compare distances. He also argues that the UbiCollab location service should provide a basic sorting functionality to the clients, e.g. sorting on proximity to user A. In the end of the section on location service Schwarz mentions that privacy is a very big issue in this service. Data should be encrypted and National laws and restrictions must be taken into account.

Schwarz gives a very brief and general description of the location service. He gives a general idea of some of the issues that must be taken into consideration, but does not elaborate on the different solutions that are available.

In the autumn of 2004 Heitmann and Jensen did work on the location-awareness of the UbiCollab platform [HJ04]. To enable dynamic discovery and use of pinpointing devices they proposed that a position service run on the pinpointing devices so that uniform communication with the location service could be achieved. A solution where the location service has two modules, a position module and a location module was presented:

> The position module handles everything that has to do directly with position. The location module handles the mapping from position to location. The position module should support storage and maintenance of position data. It must also handle calculations on positional data such as calculating distance between two positions, sorting with regards to position and nding the device closest to a position. This module will also handle the communication with the position services.
>
> The location module should support storing and maintenance of location information with regards to a context. The context will often be bound to the collaboration instance and user. With location information we mean information that can be used to build a logical map.

In the prototype they presented only the position module of the location service is implemented. The work of this thesis will build on the work done and complete the location service so that also the location module is implemented.

3.4 Database solutions for handling location data

3.4.1 Oracle Database 10g and Oracle Spatial

Oracle Database 10g together with Oracle Spatial provides a framework for storing spatial information and deploying wireless location based services. The database solution solves key problems such as how to [Lop03]:

- Efficiently store, retrieve, and manage location data and attribute data from a single, open database.
- Improve performance for very large databases containing terabytes of location data.
- Maintain versions of spatial database tables transparently, and identify any conflicts in table values.
- Store location data once and make it accessible to users from multiple, heterogeneous GIS tools and e-Business applications without any modification via open, stadards-based APIs.

The database system supports three basic geometric forms:

- *Points and point clusters:* Can be used to represent location such as buildings, vehicles, users or devices.
- *Lines and line strings:* Lines can represent roads, railroads etc.
- *Polygons and complex polygones with holes:* Represent outlines of cities, districs, buildings etc.

Oracle Spatial has several spatial operators such as *contains*, *covers* and *any-interact* that allows us to answer specific spatial queries. The system also has special spatial indexing that helps optimize the performance of such spatial queries.

3.4.2 MapInfo SpatialWare

MapInfo SpatialWare is a tool for storing and and manipulating spatial data in a relational database. It is available in two configurations, SpatialWare for Microsoft SQL Server and SpatialWare DataBlade for Informix 9.2 Databases. To enable a relational database to handle spatial data the provision of three component parts are required [MS001]:

- Spatial Data Type defining the data structure and storage mechanism.
- Spatial Indexing providing custom index structure to handle spatial data.
- Spatial Operators extending the SQL interface to the data.

As with Oracle Spatial this system also supports three basic geometric forms:

- *Points objects:* Point is the simplest of the classifications; it is defined as a single set of x,y, (z) coordinates
- *Lines objects:* Lines are linear constructs that include simple two point lines, multi-point lines (polyline), curves and arcs.
- *Area Objects:* Area (polygon or surface) represents bounding areas such as lakes, states, cities etc.

The MapInfo SpatialWare provides more than 150 spatial operators which can be classified in the following categories: constructor, general, measurement, observer, spatial and spatial predicate. The most interesting spatial operators are measurement operators such as *HG_DISTANCE* and spatial predicates such as *ST_OVERLAPS*, *ST_CONTAINS* and *ST_ADJACENT_TO.*

3.4.3 Discussion

As we can se from the above presentation of two different database systems for handling spatial data the functionality they provide are very similar. Both have a framework for storing spatial data in special data types. They provide an extension to SQL so that spatial queries can be created with the help of spatial operators. There are of course many more systems that could be used, but with regards to the need of this thesis they provide much the same functionality.

Chapter 4

Analysis and design

In this chapter I will look at the different issues concerning the Location service in UbiCollab and how they can be solved. The result of this is a design scheme that will be used when building the Location service in the UbiCollab prototype.

4.1 The Location service in relation to other Services

As anticipated in section 2.1 in UbiCollab there are three services that will have the give the main functionality when it comes to location awareness. The services are Collaboration service, Location service and Position service. To support the location awareness a three-level approach will be used where the different services handles different concers.

4.1.1 Position service

It is a service that runs independently on each pinpointing device. It enables the location service to have a standard interface to interact with when collecting the position data.

4.1.2 Location service

The location service consists of two modules: position module and location module. The main goal of the service is to provide position and location information for the other services in the UbiCollab platform.

Position Module

The position module shall handle everything that has to do with simple position. Position gives a universal specification (i.e. coordinates) of where an object is independent of context. This level must provide basic services such as:

- Comparison of position
- Dynamic discovery of pintpointing services (using directory service and resource collector

The pinpointing devices are regarded as resources that can be discovered and used by UbiCollab. So the Position module must be able to discover pinpointing devices as the environment change and add and remove devices to the list of available pinpointing devices accordingly. To simplify the communication between the position service and a 3rd party positioning/pinpointing system, these 3rd party systems must implement a standard interface (position service).

The position module have a table in the database where it stores position data collected from the various pinpointing devices.

Location Module

This is the level where the actual conversion from position to location takes place. The location module will get position information from the position module. Then it uses a location database and spatial queries to convert the position information into location information. The location module must provide more advanced services than the position module such as:

- Advanced location model for representing location
- Answer spatial queries
- Build logical maps

To do this the location module must build up a location database containing information on the various locations. The location module will not store information on where a device is at a given time, rather get this information from the position module and convert it to find the location. How the database is developed is described in the section “Location database and spatial Queries”.

4.1.3 Collaboration service

This level is the highest level of abstraction and the collaboration service must be designed to support the contextualization of the map created by the location service. It must be able to provide the following information to help contextualize:

- Collaboration instance information
- Status of users and resources
- Presence of involved users
- Profile and preferences of users'

The collaboration service must also provide the possibility to annotate maps. These annotations are specific to each collaboration instance and will help support the transition from space to place, by enabling each collaboration instance to give special meaning to the spaces represented in the map.

4.2 Location database and spatial queries

To be able to store and access it, the location information needs to be stored in a database scheme. There are two possible solutions for storing and maintaining this data. The first solution is to use a regular database to store the data and build all the functionality needed on top of this. However another solution is to use a database tool that is designed for handling spatial data. This will simplify the process of building logical maps and answering spatial queries since much of the functionality needed can be found in such a tool. The disadvantage of using a third party solution is that the service then becomes dependent on running on a system that can support this solution. Also a third party database system with support for spatial data usually requires a lisence for use and this can increase the cost of running the platform.

In the previous chapter on related work Oracle Database 10g combined with Oracle Spatial and MapInfo SpatialWare were presented as such solutions. Both solutions provide the needed functionality for the location module so either solutions could be chosen. Because Telenor has an Oracle database with Oracle Spatial available this solution was chosen for the prototype. It provides the necessary tools for storing the data and also functionality for searching through the data based on spatial parameters. It has several spatial operators such as *contains*, *covers* and *anyinteract* that allow us to answer specific spatial queries.

To be able to use the location module a database of information on locations must be designed. Because the position module works with the WGS84 coordinate system, this will also be used as reference system in the location database. The location database must be able to store information on a wide range of location types. A location can be a room, a building, a city etc. Locations can be defined in 3 dimensions (x,y,z coordinates) or just 2 dimensions (x,y coordinates). When a location is defined in only 2 dimensions the location module will not differentiate between a device located on the second floor or on the first floor of a building defined only by its area.

4.2.1 Datamodel for location database

In Oracle Spatial there is a mechanisms for representing geometry:

- *Object-relational model:* uses a table with single column of type MDSYS.SDO_GEOMETRY and a single row per geometry instance.

What this means is that the relational-model uses the existing numeric SQL-types for geometry storage and the object-relational model uses defined SQL with Geometry types. This means that the object-relational model has implicit support for a range of geometry types such as arcs, circles, compound polygons etc. With the object-relational model it is easier to create geometries and therefore this is the natural choice for this solution.

In Oracle Spatial the data model is a hierarchical structure consisting of elements, geometries and layers, with layers being the topmost abstraction. In this thesis elements and geometries will be used to create the locations. Each location will be modelled as a geometry which is in turn made up of one or more elements. A further technical description on how to create locations as a geomtetry object will be presented in the Prototype chapter.

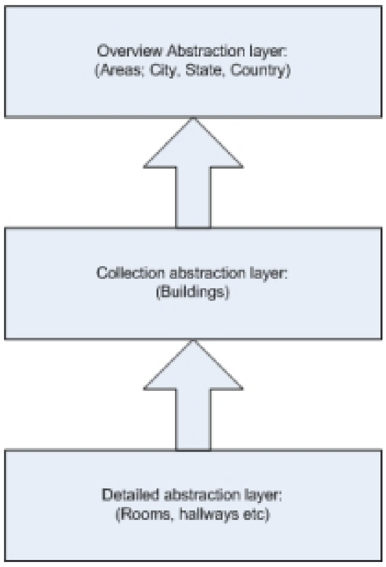

Overview Abstraction layer:
(Areas; City, State, Country)
Collection abstraction layer:
(Buildings)
Detailed abstraction layer:
(Rooms, hallways etc)

Figure 4.1: Datamodel layers

One idea on how to use layers in the solution is to have different layers for different location abstraction levels see Figure 4.1. The lowest layer could be single locations such as rooms, then a layer for collections of locations such as buildings and then the topmost layer would be areas such as city, state etc. This way when one is searching to find a given location one can start by searching through the low level, if no such location is found at the position the search continues but on a higher abstraction layer.

Each location will be given an unique ID to identify it. This ID will be a String such as "NTNU-Gløshaugen-IT-bygget" that clearly identifies the location. In different contexts the name or label on a location will vary, this will be handled by having a separate data source over the locationIDs, their corresponding labels and contextIDs. When a location is found the location module will look up the location in the label database and find the label that is used with the context. If no label is assosiated the ID of the location will be returned, therefore the locationIDs should be as informative as possible. The service that is using the location service must therefore specify a contextID in its queries if it wants to get a label on the location not just the ID String. This will be very useful when using the information to create logical maps since each location can be labeled with different labels given different contexts.

4.3 API design

In this section a simple API that is designed for the location service is presented. Each function is described with details on input parameters, output parameters and how the function works. All of these functions will not necessarily be implemented in the prototype presented in the next chapter, but they should be provided in a finalised version of the location service. The full XML description of the location service is included in appendix B.

getLocation:

- Input: position
- Output: location
- Description: returns the location of a given position. Does not consider context and the return value is the ID-string of the location.

getDeviceLocation:

- Input: entityID, requestEntityID, contextID
- Output: location
- Description: returns the location of a given entity. When the location is found in the location database, the function then searches to see if it can find a label for the location for the given context. If a label is found

the label is returned, if no label is found or no contextID is given, the ID-string of the location is returned.

getUserLocation:

- Input: userID, requestEntityID, contextID
- Output: location
- Description: returns the location of a given user. The function must find which device the user is currently using or used most recently and then return the location of this device. As with the getDeviceLocation function it returns the label associated with the context or the locationID if no label is found.

getDeviceListLocation:

- Input: locationID, deviceType, requestEntityID
- Output: deviceList
- Description: returns a list of all devices of the given deviceType present in the given location. If no deviceType is specified the function will return all the devices in the given location.

getLabel:

- Input: locationID, contextID
- Output: label
- Description: returns the label that matches a locationID given a specific context. If no label can be found the function will return the locationID.

addLocation:

- Input: locationID, perimeterPoints
- Output: output
- Description: This is a helper method to add a location to the location database. The locationID is an ID-string which clearly and uniquely define the location such as “NTNU-Gløshaugen-IT-bygget”. The perimeterPoints define the outline of the location. The perimeterPoints parameter must be given in the following format: each point is separated by a | and the X and Y value of a point is separated by a #. Also to create a closed polygon (all locations must be polygons) the first and last perimeterPoints must be equal and the outline be defined in a counterclockwise order. The function returns a boolean value based on the success of the insert.

addLabel:

- Input: locationID, contextID, label
- Output: output
- Description: This is a helper method to add a label to the location database. The locationID and contextID is the primarykey which define the situation for the label. So in the label database table the locationID and contextID pairs must be unique.

There are still some issues that are not handled by this simple API because they require a more complex functional structure. One issue is how to create and distribute logical maps. The next section will look at some solutions within the database scheme that has been presented.

4.4 Logical maps

In the related work chapter different representations for locations where presented. In the Oracle database scheme presented in the previous section the locations are stored in a geometric coordinate model. So creating a map of a specific area or containing specific locations is quite feasible. Creating a more logical map which bases its mapping on hierarchical and spacial relationships is also feasible but will require more work on transforming the location data.

To create a map of a specific area one can use the features in Oracle spatial to get all the geometries (locations) within the area and then draw the map with specific labels to create a normal graphical visualisation of the locations. However for some applications/devices such a visualisation might not be implementable. For devices where this visualisation does not fit a logical map might be the right tool. To create a map of how different locations relate without using a graphic visualisation, the spatial relationsships between the various locations must be deduced or explicitily modeled. In the datamodel presented in the previous section only the geometric properties of the location was modeled. However it is quite feasible to extend the model so that also spatial relationships such as connectivity and containment can be explicitily modeled to get a hybrid model as described in [BD04].

A possible design for this hybrid model would be to add extra database tables for each spatial relationship that needs to be modeled. For instance modeling connectivity can be done quite easily by adding a table that contains one entry for each connection between two locations. To enable the creation of big logical maps, using the hybrid model based on a hierarchical symbolic model has many advantages. The hierarchical model is based on modeling the containment relation explicitly in the model. To adapt the database scheme presented in the previous section each location must store atleast which location it has as a parent (directly above it in the containment-hierarchi). A diagram showing how this hybrid model would work is presented in figure 4.2

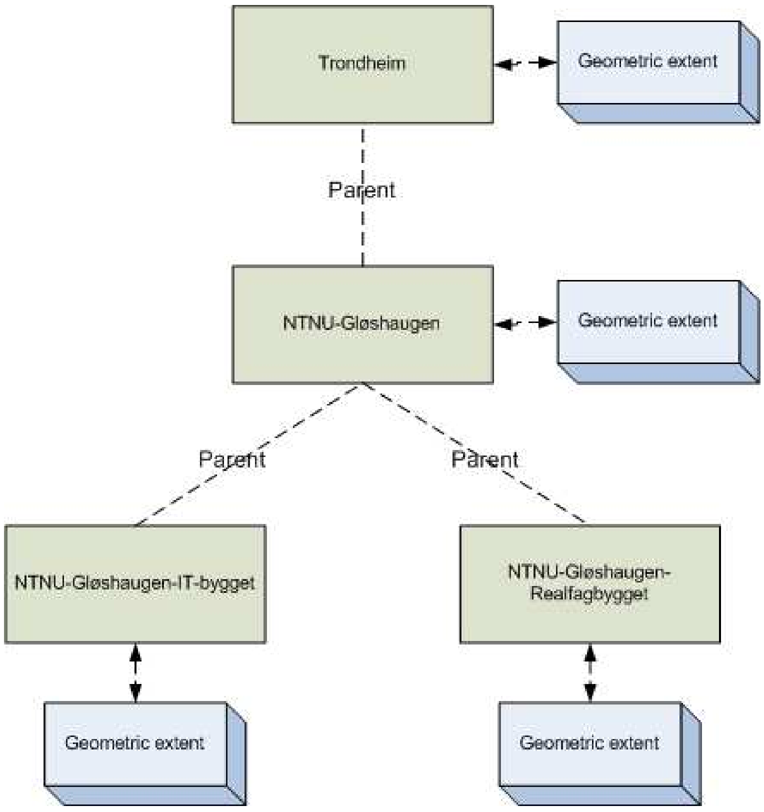


Figure 4.2: Hybrid location model

Chapter 5

Prototype

The prototype developed with in connection to this master thesis is a proof of some of the concepts discussed in the previous chapters. The focus for the prototype is on the location service of the UbiCollab platform. It is based on the prototype developed by Heitmann and Jensen in the autumn of 2004 [HJ04]. The main goal for the development is to complete work the location service so that it can handle requests for both position and location data.

5.1 UbiCollab overview

In this section an overview of the UbiCollab platform prototype developed by Heitmann and Jensen is presented. Special attention is given to the location service where most of the work on this prototype will be done. Also some issues regarding existing problems with the prototype are disscussed.

The UbiCollab platform consists of several independent UPNP enabled services. This means that the services can be dynamically discovered by other services and services can communicate using their published service description. An overview of the different services implemented is presented in Figure 5.1 which show the package diagram for the UbiCollab java prototype. Each service is implemented as its own package and in addition there are some administrative packages (util and common). Each of the services have their own startup file and can be started independently.

The location service consists of two modules; position module and location module see Figure 5.2. The work on this prototype will complete the location module so that the location service can handle queries about location. The location service consists mainly of the locationDevice class which makes the service UPNP enabled and the two modules that answer the position and location queries. The way the service is built the position module has its own dataset for storing

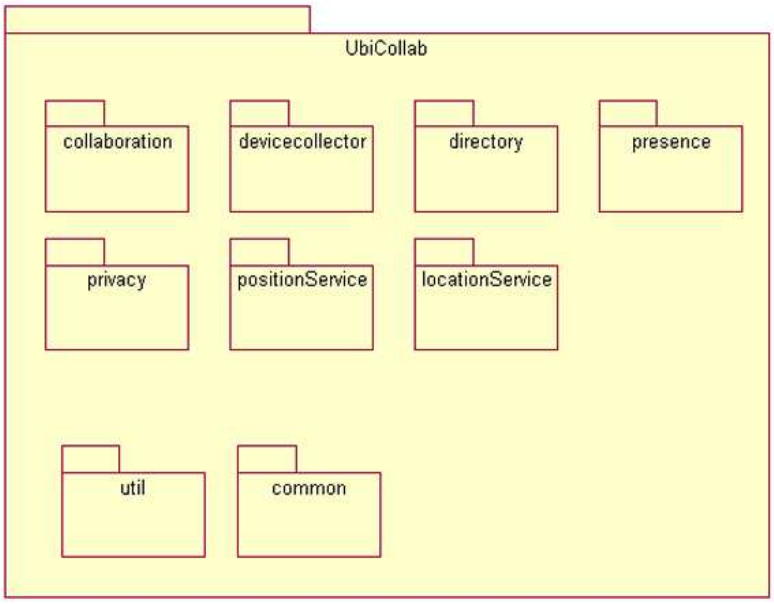


Figure 5.1: UbiCollab java-package diagram

position information which is created in the same mysql database that the other services use to store device and user information. When a request for position information is sent to the position module it will search through its database to see if it has valid position information, if the information is not found in the database, it will query all the position services for the information.

One of the issues with the existing prototype is that there is no direct link between a user and the devices registered with the platform. For instance when logging on to the platform through the PDA-client the prototype does not register the PDA used as a device, only the user session is activated. This loose connection between users and devices is a problem when trying to find the position/location of a user since the answer often is dependent of which device the user is using.

There is also the issue of how the devices register with the platform. The devices are registered through publishing their UPNP service description (XML). However there is no clear deviceID stored with the devices when they are stored in the directory service and this makes it hard to uniquely identify devices when searching for them in the position service. The approach that was made in the autumn 2004 was to use the closest thing to a deviceID, friendlyName, which is stored with the device information in the directory service. However this requires that the friendlyName parameter is set to a different value for each device in their service description.

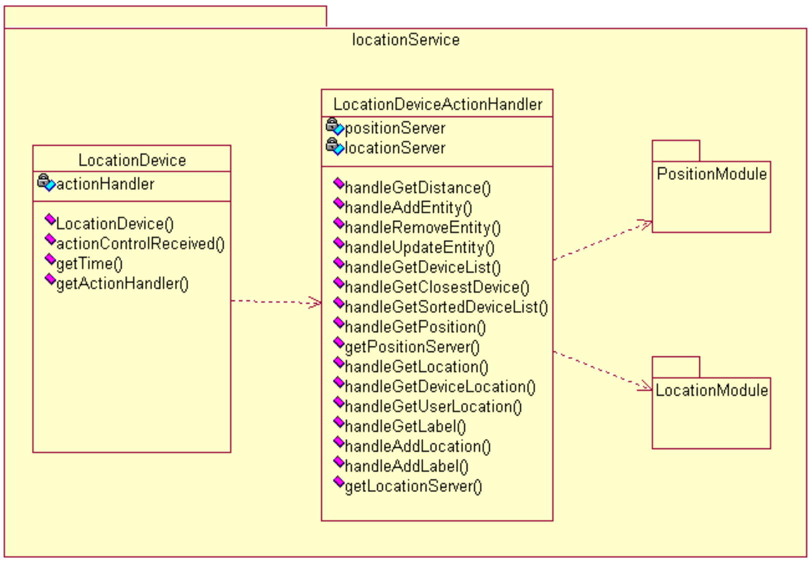


Figure 5.2: Location service class diagram

5.2 Location module

This section describes the design and implementation of the location module in the location service. To be able to store and update the location information a location database is required. As mentioned in the analyses and design chapter the choice of database system fell on Oracle Spatial.

The database consists of three tables used for storing geometries, one for each layer described in the previous chapter. Each of the tables are set up with their own spatial index on the WGS84 coordinate system. This means that it is possible to perform spatial searches and queries on the tables.

To create a table that can store the location information the special Oracle Spatial datatype of MDSYS.SDO_GEOMETRY is used for storing the geometries. To organize and keep an overview of the tables containing spatial information Oracle spatial uses a special table containing geometry metadata. It is this table which will keep track of which coordinate system the data should be stored in and what level of tolerance should be used when performing spacial queries on a table. The following SQL code is used for creating the table and setting up the spatial indexing:

```
-----------------------------------------------------------------------
             Creating detail location table and index:
-----------------------------------------------------------------------
CREATE TABLE detail_location (
  location_id VARCHAR2(32) PRIMARY KEY,
  shape MDSYS.SDO_GEOMETRY);

INSERT INTO USER_SDO_GEOM_METADATA
  VALUES (
  'detail_location',
  'shape',
  MDSYS.SDO_DIM_ARRAY(
    MDSYS.SDO_DIM_ELEMENT('Longitude', -180, 180, 0.5),
    MDSYS.SDO_DIM_ELEMENT('Latitude', -90, 90, 0.5)
     ),
  8307   -- SRID for 'Longitude / Latitude (WGS 84)' coordinate system
);

CREATE INDEX detail_loc_spatial_idx_cs
ON detail_location(shape)
INDEXTYPE IS MDSYS.SPATIAL_INDEX;
```

The location module consists of two classes, see Figure 5.3. The LocationServer class handles all the requests from the locationDevice. The LocationData class is a wrapper class for accessing the location data in the database. To communicate with the oracle databace the class uses the standard JDBC interface with the Oracle driver.

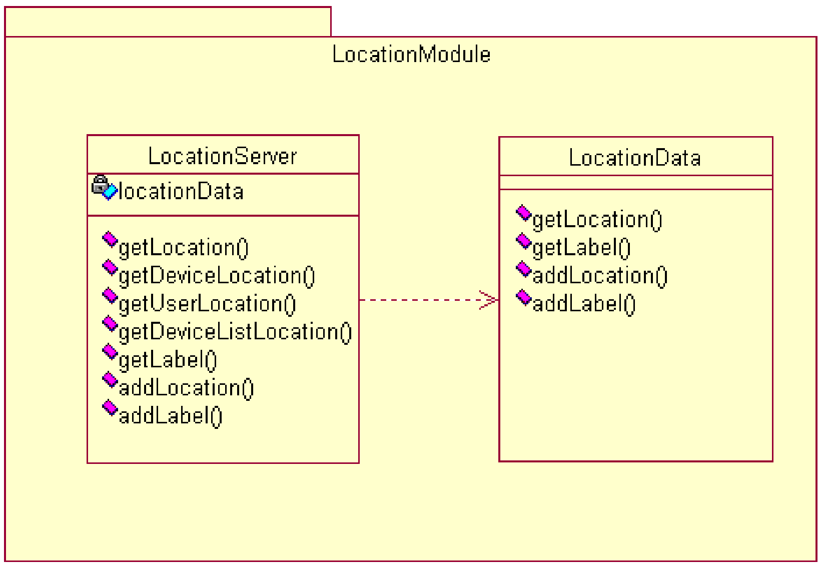


Figure 5.3: Location module class diagram

In addition to the database tables for storing the location information and their geometries there is also a table for storing label information. Each location is defined by a unique string ID which should be descriptive so that it gives an understanding of where the location is. However a location may have different names in different contexts and this is where the labels come in. This prototype enables the services using the location service to associate labels with the different locations based on which context the location is needed for. The label is stored in the table along with the locationID and a contextID. Together the locationID and the contextID must make a unique pair. When a service retrieves a location for a specific context it can associate a label with the location by using the function addLabel(). When a request for a location includes a contextID the locationModule will first find the location, then it will search to see if it has a label for the specific locationID and contextID pair. If a label is found the label is returned, and if no label is found the locationID is returned.

The location module can handle queries for the label or ID of a given position, it can also handle queries for the location of a given device. The location module stores no information on the devices or users location, this is left to be handled by the position module. This is because it is the position module that handles the communication with the position service and stores the position data of devices. So when a query for the location of a specific device is received the location module queries the position module for the position of the device and then goes on to identify the location from the position data it receives.

In this prototype the position data and the location data are in separate databases, which presents some issues for some of the more advanced queries. For instance searching for all the devices in a specific location becomes very hard because the location database has no information on the position of the different devices. If the positions of devices had been plotted in the same database as the location information it would have been a simple matter of using spatial operators to find all the positions contained within the locations geometry. However because converting the position database and merging it with the Oracle spatial database would require a massive restructuring of the position module and position service this has not been done in this prototype.

5.3 Location queries

The new functionality in the location service can be accessed by other services throught the location service's UPNP device. In Figure 5.4 a simple location query from a client application to get the locationID of a given position is presented as a sequence diagram.

The client application sends an action with the name getLocation to the LocationDevice through its service interface. The LocationDevice identifies the action and retrieves the ArgumentList contained in the action object. It then sends a getLocation message to the LocationDeviceActionHandler. The ActionHandler converts the position data, which is included in the ArgumentList as

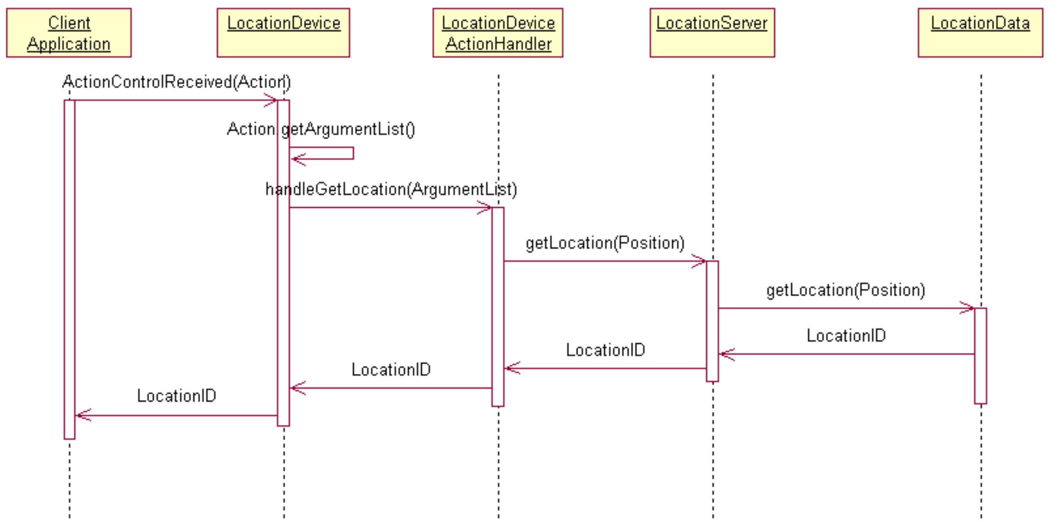


Figure 5.4: Sequence diagram for a getLocation query

a string, to a Position object and sends a getLocation message to the LocationServer with the position object as input parameter. The LocationServer then passes the message on to the LocationData object which searches through the database for the location that contains the position and returns the LocationID. The LocationID is passed back through the chain and returned as the outputArgument of the client application's query. When the LocationData object searches for the location it will first search in the lowest location layer, the "detail location layer". If no location is found it then searches the medium layer, the "collection location layer", and then goes on to search the top layer, the "overview location layer".

Another example is presented in Figure 5.5 where the sequence diagram of a query searching for the location of a given device is depicted.

The message sequence of the search for device's location is similar to the sequence for finding the location of a given position, but it also involves contextID, locationLabel and the PositionModule. The ActionHandler now extracts the EntityID and the ContextID from the argument list and sends the getDeviceLocation message with these as input parameters to the LocationServer. The locationServer then query the PositionModule for the position of the device with the given EntityID. When the position of the device has been found the location can then be found by querying the locationData for the location matching the position. Once the location has been identified the LocationID and the ContextID can be used to find and return the LocationLabel for the given context. If no contextID is included in the query or no matching label can be found the LocationID will be returned as the outputArgument of the query.

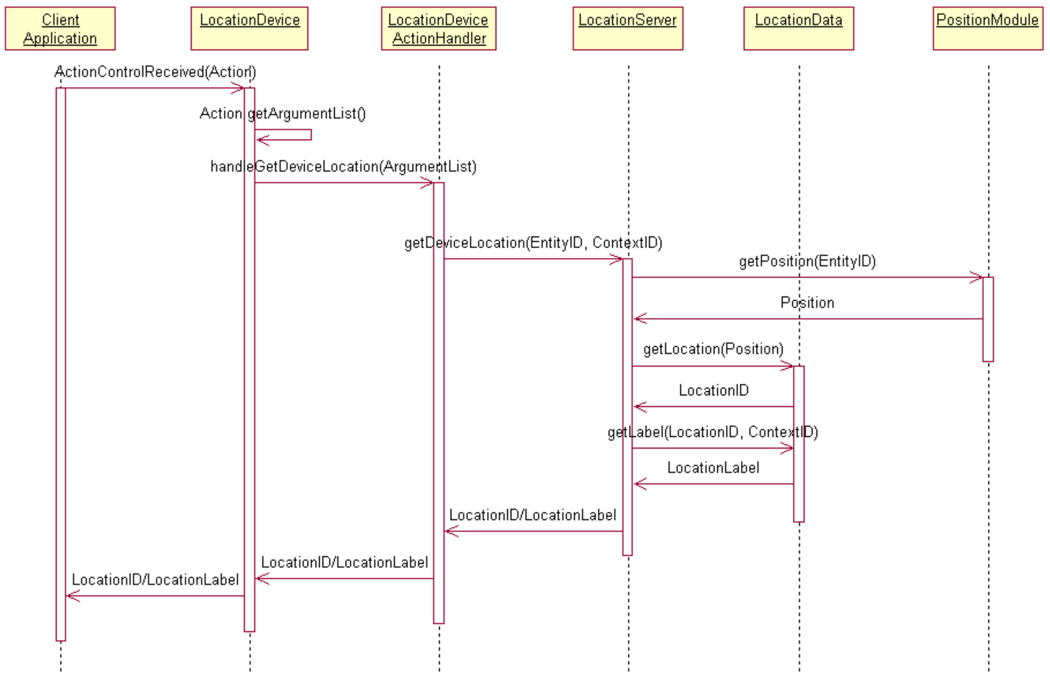


Figure 5.5: Sequence diagram for a getDeviceLocation query

Chapter 6

Evaluation

In this chapter an evaluation of the theoretical and the practical work done on this master thesis is presented. First an evaluation of the prototype is done. Then the teoretical solutions presented in the analysis and design chapter are evaluated. A short evaluation of the work process throughout the master thesis is also inlcuded.

6.1 Prototype

This section starts of with a demonstration of how the prototype can be used to provide other services with location information. Then an evaluation of the prototype's limitations and future possibilities is presented.

6.1.1 Demonstrator

The original plan for the demonstrator was to use the shared display application developed by Heitmann [Hei05] to demonstrate the use of the location service, however because of time limitations and latent restrictions in the UbiCollab prototype this has not been achieved. One of the limitations is that there is no direct connection between the user and their devices in the UbiCollab prototype and to rectify this the whole prototype would have to be altered and partially redesigned. So the demonstration of the prototype will be in the form of a set of test data entered into the database and test queries to find the location of devices and positions. The aim of the test is to show how the location service now can handle location queries and also add new location information when the service is running.

To simulate that there is a service sending requests to the location service I have created a test class which emulate a service by invoking actions on the

LocationDevice through the CollaborationUPNPControlPoint as a real service will do.

Inserting test data

To test the location service the database have to be filled with a set of locations and I have chosen to model some locations in Trondheim. The full test-set is included in appendix C and will not be presented in this section however to demonstrate how the location service works parts of the test-set will be presented.

The first test for the location service is to add a location to the database. This is done by invoking the "addLocation" action. The addLocation action requires two input parameters, locationID and perimeterPoints. The perimeterPoints is a collection of coordinates that define the outline of the location in a counter-clockwise order. The code for inserting a location into the location database is as follows:

```
--------------------------------------------------------------------------
               Adding location to location database:
--------------------------------------------------------------------------

Hashtable arguments = new Hashtable();
arguments.put("locationID", "NTNU-Gløshaugen-IT-bygget");
arguments.put("perimeterPoints",
    "63.417387678888889 # -10.401984163888889 | " +
    "63.417558822222222 # -10.402812941666667 | " +
    "63.417652011111111 # -10.402710913888889 | " +
    "63.417566277777778 # -10.402239972222222 | " +
    "63.417745441666667 # -10.402051827777778 | " +
    "63.417662763888889 # -10.401648816666667 | " +
    "63.417387678888889 # -10.401984163888889");
result = collaborationUPnPControlPoint.invoke(locationDevice, "addLocation", arguments);
```

The next test is to add some label data to the database so that the locations are given different labels for different contexts. The code for inserting a label into the database is given below:

```
--------------------------------------------------------------------------
               Adding label to location database:
--------------------------------------------------------------------------

Hashtable arguments = new Hashtable();
arguments.put("locationID", "NTNU-Gløshaugen-IT-bygget");
arguments.put("contextID", "Prosjekt");
arguments.put("label", "Kontoret");
result = collaborationUPnPControlPoint.invoke(locationDevice, "addLabel", arguments);
```

A device with a specific position will also be added to the position database so that all the functionality can be tested:

```
--------------------------------------------------------------------------
               Adding device to position database:
--------------------------------------------------------------------------

Hashtable arguments = new Hashtable();
arguments.put("positionX", "63.417498650000000");
arguments.put("positionY", "-10.402015497222222");
arguments.put("height", "0");
arguments.put("timeStamp", "0");
arguments.put("timeToLive", "0");
arguments.put("entityID", "testDevice");
result = collaborationUPnPControlPoint.invoke(locationDevice, "addEntity", arguments);
```

The addLocation, addLabel and addEntity actions return a boolean value based on the success of the insertion. For the rest of the test location data see appendix C.

Testing location queries

Now that some test data has been inserted into the database it is possible to test to see if the location queries run as they are intended to run. For the queries I have run I have used the test data from appendix C and each query is presented with the description of how to run it and the results for different values.

The first test is the simple getLocation query:

```
--------------------------------------------------------------------------
               Running getLocation query:
--------------------------------------------------------------------------

Hashtable arguments = new Hashtable();
arguments.put("position", "63.417498650000000#-10.402015497222222");
result = collaborationUPnPControlPoint.invoke(locationDevice, "getLocation", arguments);
```

This query does not take into consideration the context and labels and therefore it returns the location, from the most detailed layer possible, that contains the position. The results from the query was very good. When trying to locate a position that was covered by a location in the detail_location layer such as "NTNU-Gløshaugen-IT-bygget" the query returned the correct locationID as the result. Also when searching with a position argument that is not in the detail_location layer the location containing the position from the closest layer above was returned. When searching for a position that is not covered by any test location the query returned "no location found error".

The next query to be tested is the getDeviceLocation action. This action requires the entityID of the device and the contextID of the context as input:

```
---------------------------------------------------------------------------
               Running getDeviceLocation query:
---------------------------------------------------------------------------

Hashtable arguments = new Hashtable();
arguments.put("entityID", "testDevice");
arguments.put("contextID", "Prosjekt");
result = collaborationUPnPControlPoint.invoke(locationDevice, "getLocation", arguments);
```

This query uses both the positionModule and the locationModule of the location service. When the location service receives a request for the location of a device it first uses the positionModule to find the position of the device and then it uses the locationModule to get the locationID or label. This query was tested with different positions for the testDevice and different contextIDs. All the tests gave the correct/expected results.

6.1.2 Evaluation and future work

The prototype has successfully implemented a locationModule that can handle simple location queries. The foundation for an advanced location service has been implemented and it is possible to extend the prototype to handle more advanced location queries such as the ones discussed in the end of the analysis and design chapter.

The prototype demonstration could not be integrated as well with the work of Heitmann as planned because the issues discussed in the previous section. This has done that the demonstration does not include any screenshots or other visual aids. Still I feel that the demonstration shows how the location service works and that it now can handle location queries in addition to position queries.

One of the major drawbacks that was encountered during the prototype development was that there are several latent restrictions in the UbiCollab platform prototype. One of the things that should be looked into in future work is to restructure and redesign the whole prototype to get a more coherent prototype and make it easier to maintain and modify.

For the location service part of the prototype it would be crucial to get a stronger link between the users and their devices so that this can be utilised in getting location information on devices, users and resources.

Another thing that should be looked into is to integrate the two databases, position and location. By having one coherent database more advanced location queries with regard to devices and users can be created.

6.2 Evaluation of theoretical solution

In the analysis and design chapter a solution for storing and querying spatial data in the UbiCollab platform was presented. One of the aims of this master thesis was to present an API for the location service in the UbiCollab platform. The design chapter presents an API that handles the basic location queries and insertion of location information. The API was used as the interface the prototype locationService should provide and the solution seemed to work well in practical work as well as in theory.

The storing of location information is one of the main issues of this master thesis. The solution presented in the analysis and design chapter is based on using a third party solution for handling the storing and querying. This has both positive and negative ramnifications. By using a third party solution the location service then becomes dependent on a specific software, that is for the service to be able to run it must be able to run an Oracle Spatial database or be able to communicate with one. However there are also positive sides such as full support for spatial indexing and querying, built in geometric types, multiple spatial operators and filters.

Another aspect of the storage and querying of location information is how the information is stored in the database. The solution presented in this thesis is based on the hybrid location model architecture presented in [BD04]. This solution gives a model that can be very rich on information and therefore answer all the important spatial queries. However the richness of the model also makes it more work intensive to create the different locations in the model.

6.3 Work process

When starting out on this master thesis I made a general plan for how much time should be spent on the various parts of the thesis work. As in many projects some time limits were exceeded an the plan had to be adjusted. All in all the plan worked out, but I should have scheduled some more time for development to be able to create a mutual demonstration with the other groups working on the UbiCollab platform.

As for the collaboration with the other groups working on the platform it has been good throughout the process. Because we decided to produce our own separate demonstrations the collaboration has been on a theoretical level. We have had meetings regularly to inform each other of how the work is going and shared information that affect the work of other groups.

Chapter 7

Conclusion

In this thesis I have studied how to handle location information. I have continued the work of Heitmann and Jensen on the location service in the UbiCollab platform by making it location-aware. This work has made the UbiCollab platform location-aware with regards to both position and location. The work has been done on state-of-the-art concepts and given a contribution to the UbiCollab project.

In the thesis I used a scenario to specify the issues and goals with regards to the work on location-awareness in UbiCollab. The goals were focused on issues on the platform level, but also more general issues with regards to handling location information has been kept in mind while working on this thesis.

The related work presents background information to understand the different issues surrounding storing and handling of location information. This understanding has been used for analysing the issues and designing the solution for a fully location-aware location service. The prototype developed in this master thesis partly implemented this design and enabled the location service to handle some of the location queries.

7.1 Summary of contributions

In this master thesis I have made the following contributions to the field of location-awareness and the UbiCollab platform:

- Presented research in the field of location-awareness and handling of spatial information.
- Created a location model for use in the UbiCollab location service.
- Created an API for the location service in the UbiCollab platform.

- Designed a complete location service which can handle both position and location information.
- Designed and implemented a prototype that shows how location information can be stored and accessed in the UbiCollab platform.
- Demonstrated through this prototype that the theory presented can be realised in the platform.

7.2 Future work

For the future there are several issues with regards to the UbiCollab platform in general and the location service in particular that could be looked into.

Because work on the UbiCollab platform has been done by several different groups over the years the platform design has slowly become less coherent. The design started out as a clear and robust design with good structure. As the different work on several of the services has been added to the design it has become less clear. Especially for the prototype of the platform this constant adding has cluttered the design and made the prototype more error prone. So an effort should be put into cleaning up the design and incorporating the different ideas that has been presented over the years in a new design that is more coherent. A new platform prototype that can show all the work done on privacy, shared display and location-awareness should be developed.

The work done in this master thesis on the location service has made it possible to handle location information. However there are several issues left to be handled. This thesis has described some of the issues regarding more advanced spatial queries such as creating logical maps. Building a location service that can handle the more advanced queries have been left for future work.

Bibliography

[Bak04] Anders Bakkevold. A shared display system for a ubiquitous computing environment. Master's thesis, Norges Teknisk-Naturvitenskapelige Universitet, 2004.

[BD04] Christian Becker and Frank Dürr. On location models for ubiquitous computing. *Personal Ubiquitous Computing*, 3(9):20–31, August 2004.

[CS02] Holger Christein and Peter Schulthess. A general purpose model for presence awareness. In J. Place, P. Kropf, P. Schulthess, and J. Slonim, editors, *4th International Workshop, DCW 2002 Sydney, Australia, April 3-5, 2002 Revised Papers*, volume 2468 of *Lecture Notes in Computer Science*, pages 24–34. Springer-Verlag Berlin Heidelberg, 2002.

[DFJ05] Monica Divitini, Babak A. Farshchian, and Børge S. Jensen. Ubicollab: Improving collaboration with location services. 2005.

[DFS04] Monica Divitini, Babak A. Farshchian, and Haldor Samset. Ubicollab: Collaboration support for mobile users. In *Proceedings of the 2004 ACM symposium on Applied computing.* ACM, 2004.

[DRD$^+$00] Alan Dix, Tom Rodden, Nigel Davies, Jonathan Trevor, Adrian Friday, and Kevin Palfreyman. Exploiting space and location as a design framework for interactive mobile systems. *Transactions on Computer-Human Interaction*, 7(3):285–321, September 2000.

[Gon04] Pedro André Cravo da Silva Garcia Gonçalves. Ubiclient: a mobile client for an ubiquitous collaborative environment. Master's thesis, Norges Teknisk-Naturvitenskapelige Universitet, 2004.

[HD96] S. Harrison and P Dourish. Re-place-ing space: the roles of place and space in collaborative systems. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work.* ACM, 1996.

[Hei05] Carsten Andreas Heitmann. Discolab: a toolkit for development of shared display systems in ubicollab. Master's thesis, Norges Teknisk-Naturvitenskapelige Universitet, 2005.

[HJ04] Carsten Andreas Heitmann and Børge Setså Jensen. Location-aware service for the ubicollab platform. Master's thesis, Norges Teknisk-Naturvitenskapelige Universitet, 2004.

[JS02] Changhao Jiang and Peter Steenkiste. A hybrid location model with a computable location identifier for ubiquitous computing. In Gaetano Borriello and Lars E. Holmquist, editors, *UbiComp 2002: Ubiquitous Computing 4th International Conference, Göteborg, Sweden, September 29 - October 1, 2002. Proceedings*, volume 2498 of *Lecture Notes in Computer Science*, pages 246–263. Springer-Verlag Berlin Heidelberg, 2002.

[Lop03] Xavier Lopez. Oracle spatial and oracle locator. White paper, Oracle Corporation, December 2003.

[MS001] Enterprise mapping deployments: Managing spatial data in a relational database management system. White paper, MapInfo Corporation, June 2001.

[Sch04] Christian Schwarz. Ubicollab - platform for supporting collaboration in a ubiquitous computing environment. Master's thesis, Norges Teknisk-Naturvitenskapelige Universitet, 2004.

[Wei93] Mark Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36 (7):75–84, 1993.

Appendix A

Previous work on UbiCollab platform

A.1 Theoretical platform

The Ubicollab consist of several native platform services. These provide the support for collaboration in a ubiquitous computing environment. They do however not have the responsibility of providing the actual services for specific type or domain of collaboration (like instant messaging, cooperative programming etc.). That is, the platform enables support for usage and management of such services, but do not provide such basic collaborative services itself. By keeping the platform clean like this the generality of the platform is ensured and the focus can be on developing new kinds of collaboration support instead of reinventing existing collaborative services.

A.1.1 Conceptual model

To represent the abstract collaboration concept Schwarz uses a *collaboration instance*. This is an entity which is meant to capture a real world activity, or context, of collaboration between people and resources they use. A person entity represent a human participant while a resource can be a physical device or electronic information. By creating a conceptual model centered around the collaboration instance several issues can be solved fairly easy. This gives a focus point for all the services and tie the whole platform together as a coherent system. Another important aspect with this solution is that it enables resources and other entities to be connected to several collaboration instances. It also enable collaboration instances to be aggregated with others and still maintain their components.

A.1.2 Platform services

The platform consist of modular services that cooperate with each other but run independently. The most complex and advanced service is the collaboration server which provides the main functionality that enable collaboration between clients.

The directory service is another important service. It provides basic resource management and is used to provide information on which resources are available at a given time. It collects information on which resources are available from resource collectors which run on the clients. This is a good solution since it enables dynamic adding and removing of resources as the device collectors discover or lose devices.

When it comes to the location service and the privacy service little or no work has been done in developing these. Developing the location service is the main focus of this project whilst the work of Braathen and Rasmussen will focus on the privacy service.

A.2 Developed prototype

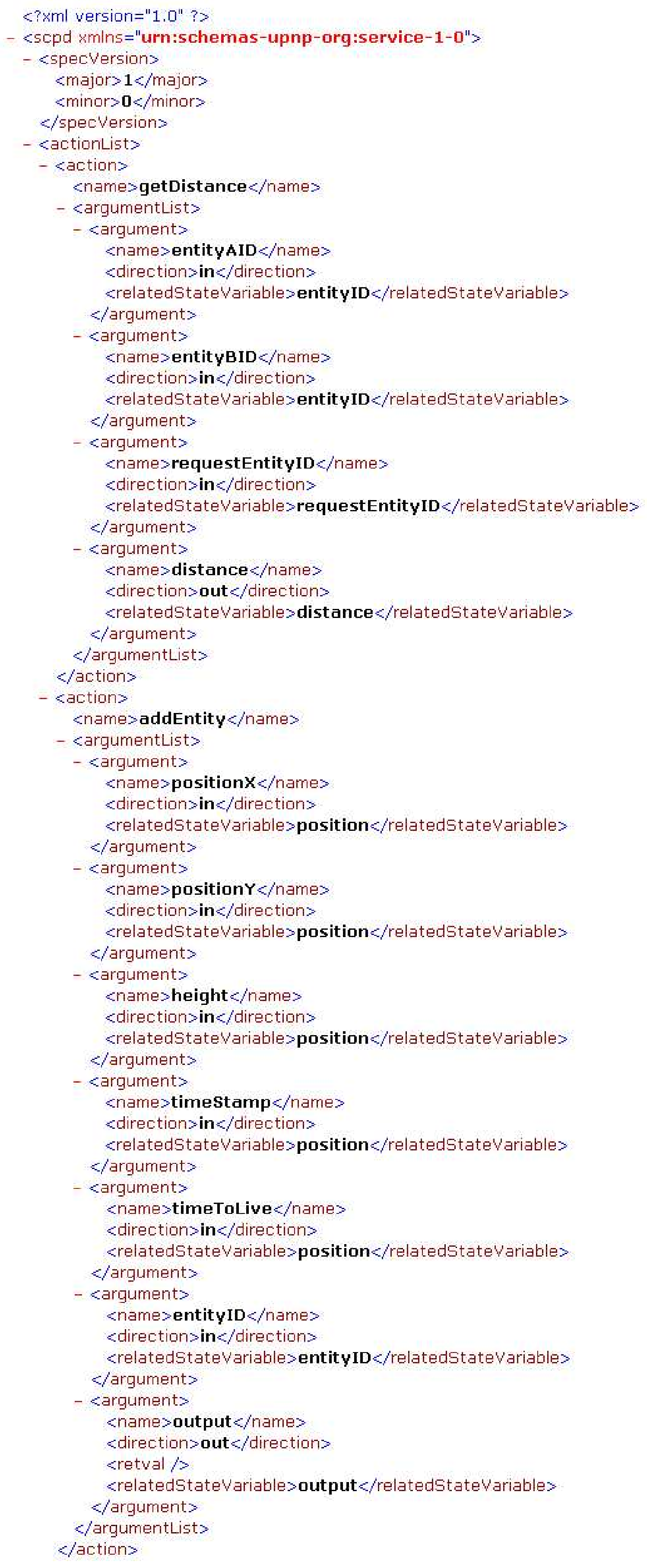
The prototype that was developed by Schwarz focused on developing the collaboration server, the resource collector, the directory service and the presence service. To communicate internally between the services in the platform UPnP is used and every service can communicate to every other service over UPnP. To communicate with the clients a web service was developed.

The prototype is well structured and implements a well arranged architecture. However there is a lack of documentation in the prototype. Almost no javadoc comments is provided and this makes the process of understanding and further developing the prototype much harder.

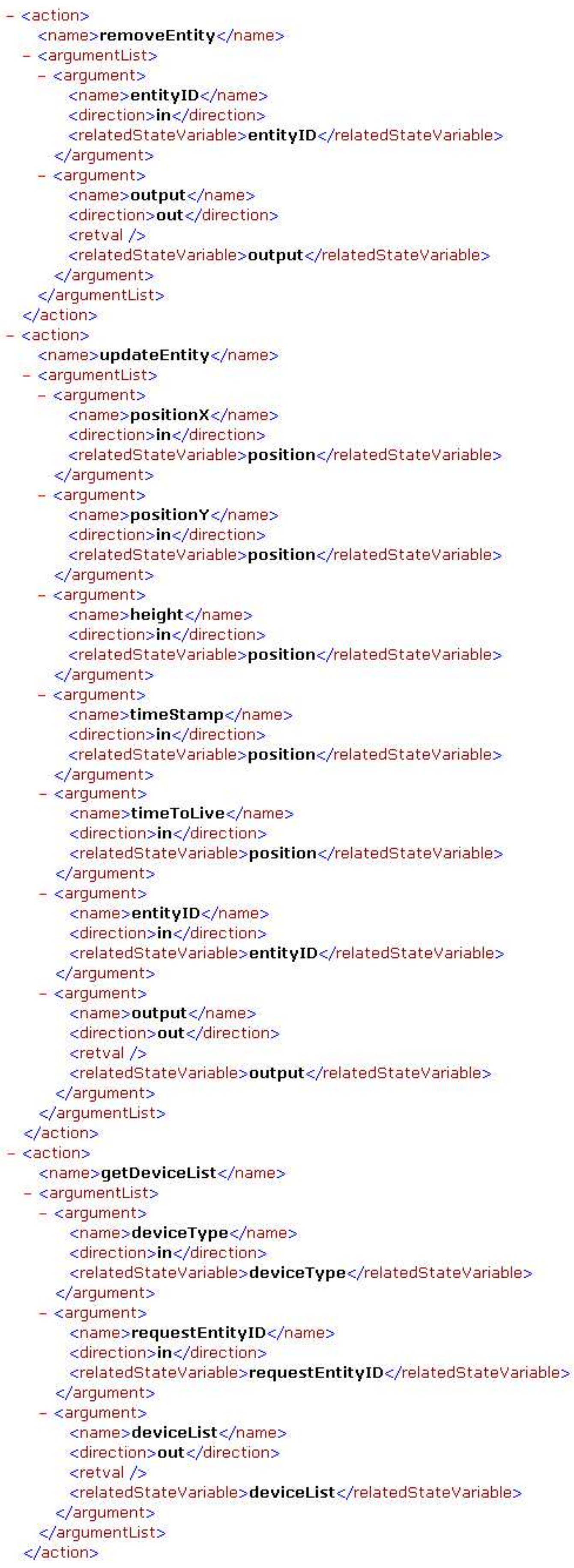
Appendix B

Location Service XML description

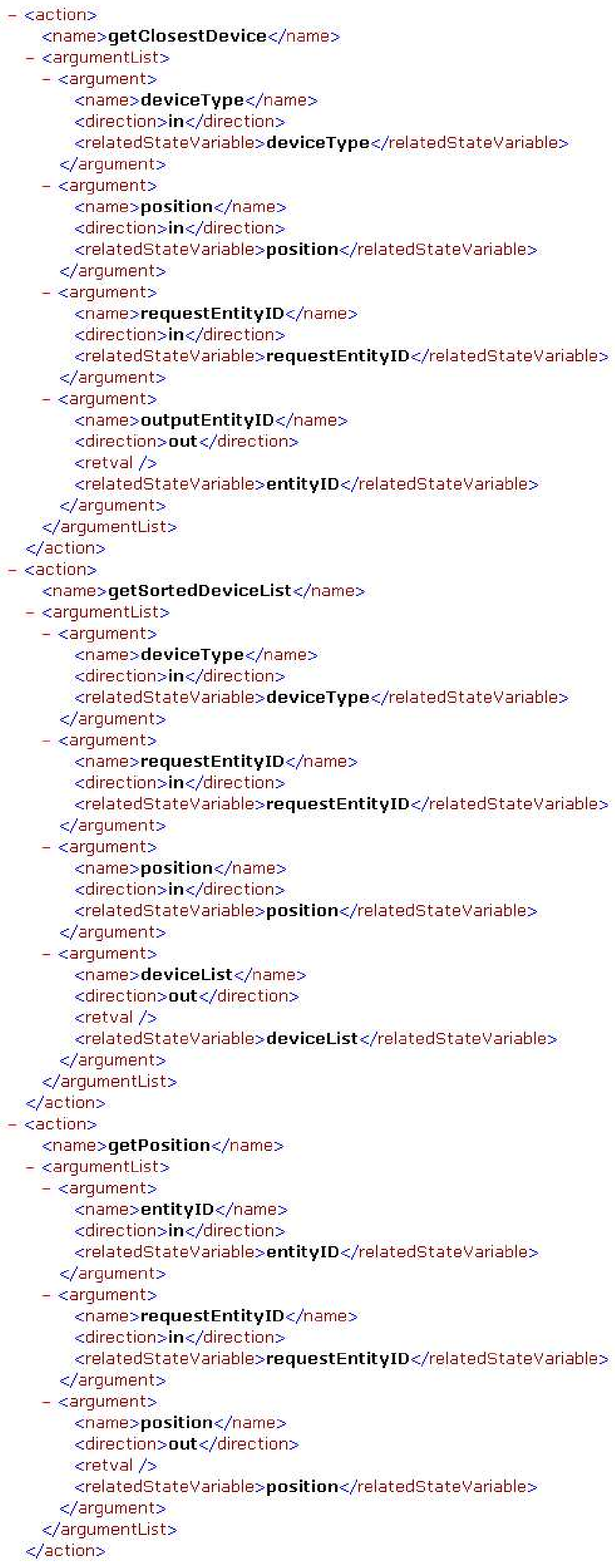
The following pages contain the xml service description for the location service. The xml-schema follows the standard UPNP service description schema.



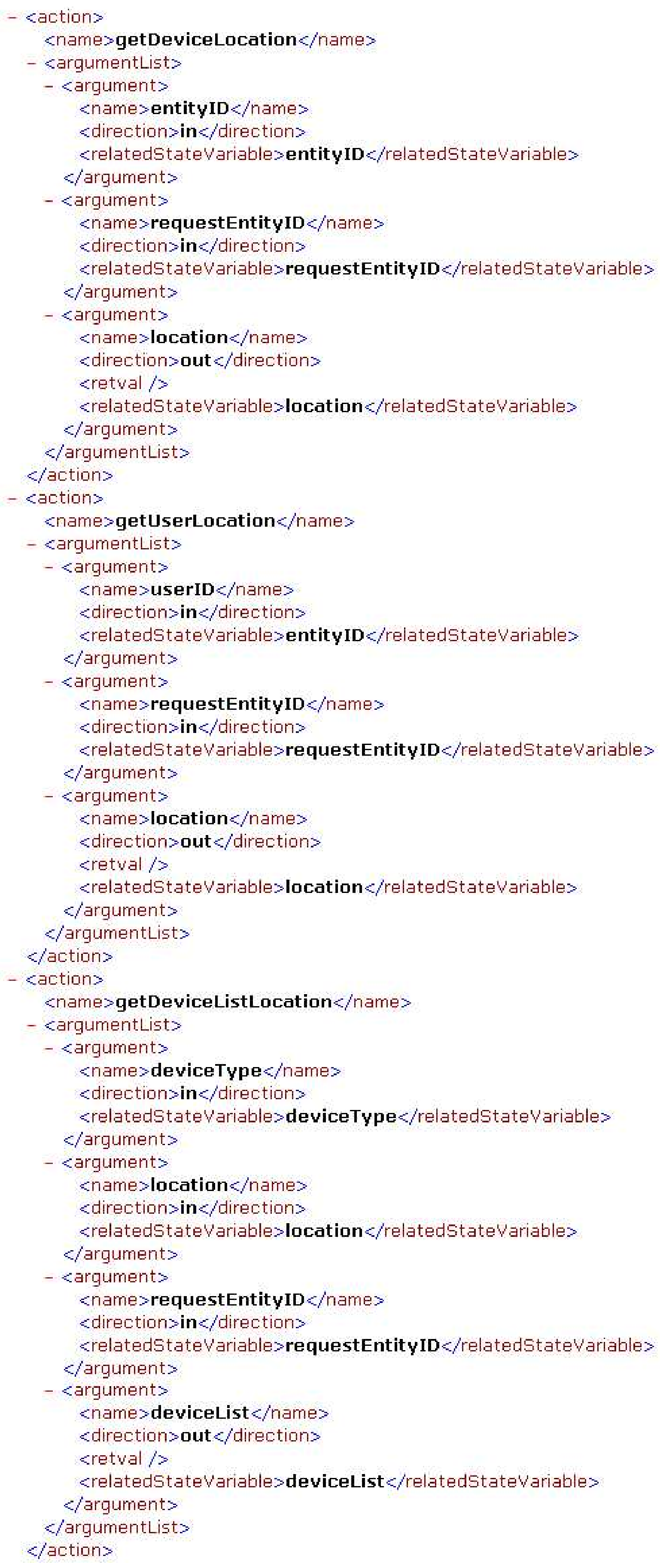
```
  <?xml version="1.0" ?>
- <scpd xmlns="urn:schemas-upnp-org:service-1-0">
  - <specVersion>
      <major>1</major>
      <minor>0</minor>
    </specVersion>
  - <actionList>
    - <action>
        <name>getDistance</name>
      - <argumentList>
        - <argument>
            <name>entityAID</name>
            <direction>in</direction>
            <relatedStateVariable>entityID</relatedStateVariable>
          </argument>
        - <argument>
            <name>entityBID</name>
            <direction>in</direction>
            <relatedStateVariable>entityID</relatedStateVariable>
          </argument>
        - <argument>
            <name>requestEntityID</name>
            <direction>in</direction>
            <relatedStateVariable>requestEntityID</relatedStateVariable>
          </argument>
        - <argument>
            <name>distance</name>
            <direction>out</direction>
            <relatedStateVariable>distance</relatedStateVariable>
          </argument>
        </argumentList>
      </action>
    - <action>
        <name>addEntity</name>
      - <argumentList>
        - <argument>
            <name>positionX</name>
            <direction>in</direction>
            <relatedStateVariable>position</relatedStateVariable>
          </argument>
        - <argument>
            <name>positionY</name>
            <direction>in</direction>
            <relatedStateVariable>position</relatedStateVariable>
          </argument>
        - <argument>
            <name>height</name>
            <direction>in</direction>
            <relatedStateVariable>position</relatedStateVariable>
          </argument>
        - <argument>
            <name>timeStamp</name>
            <direction>in</direction>
            <relatedStateVariable>position</relatedStateVariable>
          </argument>
        - <argument>
            <name>timeToLive</name>
            <direction>in</direction>
            <relatedStateVariable>position</relatedStateVariable>
          </argument>
        - <argument>
            <name>entityID</name>
            <direction>in</direction>
            <relatedStateVariable>entityID</relatedStateVariable>
          </argument>
        - <argument>
            <name>output</name>
            <direction>out</direction>
            <retval />
            <relatedStateVariable>output</relatedStateVariable>
          </argument>
        </argumentList>
      </action>
```



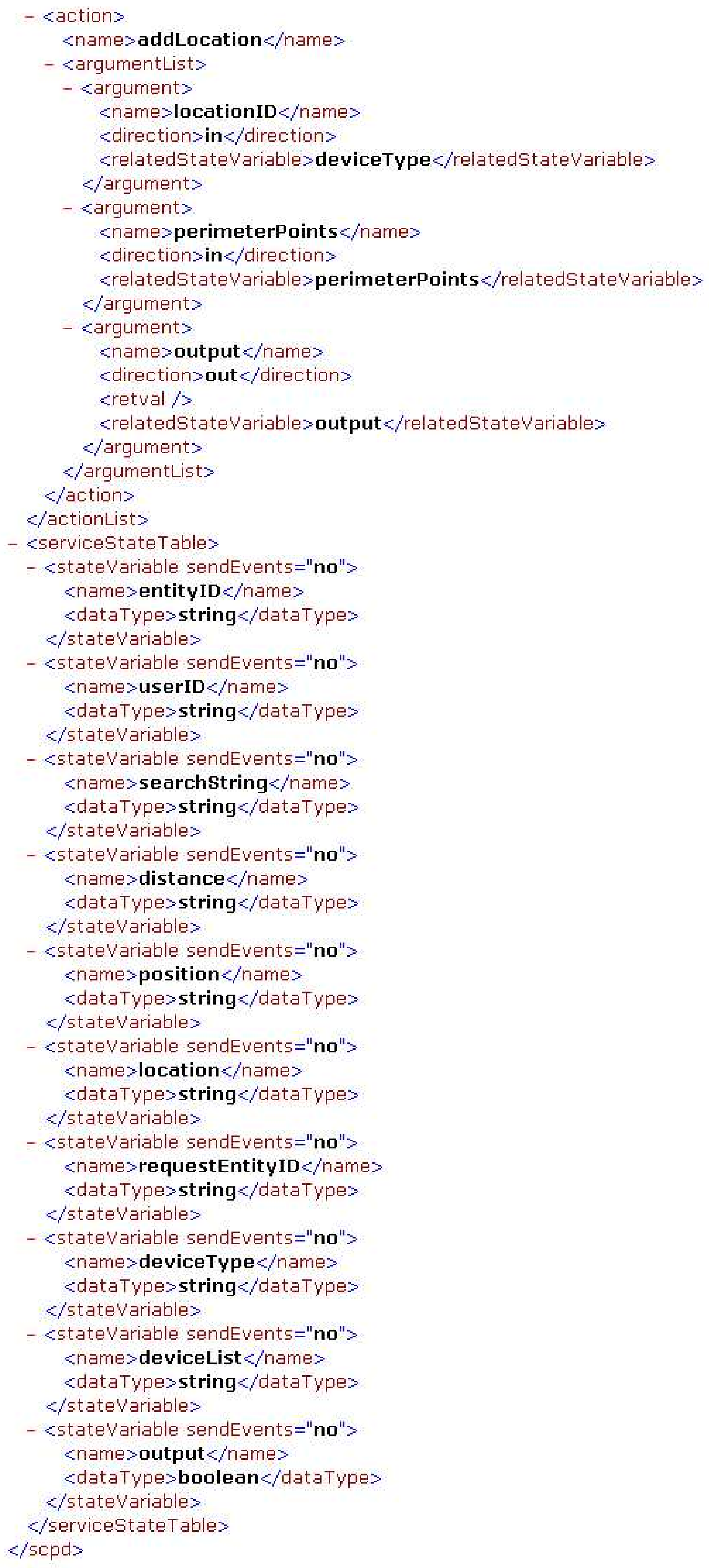
```
- <action>
    <name>removeEntity</name>
  - <argumentList>
    - <argument>
        <name>entityID</name>
        <direction>in</direction>
        <relatedStateVariable>entityID</relatedStateVariable>
      </argument>
    - <argument>
        <name>output</name>
        <direction>out</direction>
        <retval />
        <relatedStateVariable>output</relatedStateVariable>
      </argument>
    </argumentList>
  </action>
- <action>
    <name>updateEntity</name>
  - <argumentList>
    - <argument>
        <name>positionX</name>
        <direction>in</direction>
        <relatedStateVariable>position</relatedStateVariable>
      </argument>
    - <argument>
        <name>positionY</name>
        <direction>in</direction>
        <relatedStateVariable>position</relatedStateVariable>
      </argument>
    - <argument>
        <name>height</name>
        <direction>in</direction>
        <relatedStateVariable>position</relatedStateVariable>
      </argument>
    - <argument>
        <name>timeStamp</name>
        <direction>in</direction>
        <relatedStateVariable>position</relatedStateVariable>
      </argument>
    - <argument>
        <name>timeToLive</name>
        <direction>in</direction>
        <relatedStateVariable>position</relatedStateVariable>
      </argument>
    - <argument>
        <name>entityID</name>
        <direction>in</direction>
        <relatedStateVariable>entityID</relatedStateVariable>
      </argument>
    - <argument>
        <name>output</name>
        <direction>out</direction>
        <retval />
        <relatedStateVariable>output</relatedStateVariable>
      </argument>
    </argumentList>
  </action>
- <action>
    <name>getDeviceList</name>
  - <argumentList>
    - <argument>
        <name>deviceType</name>
        <direction>in</direction>
        <relatedStateVariable>deviceType</relatedStateVariable>
      </argument>
    - <argument>
        <name>requestEntityID</name>
        <direction>in</direction>
        <relatedStateVariable>requestEntityID</relatedStateVariable>
      </argument>
    - <argument>
        <name>deviceList</name>
        <direction>out</direction>
        <retval />
        <relatedStateVariable>deviceList</relatedStateVariable>
      </argument>
    </argumentList>
  </action>
```



```
- <action>
    <name>getClosestDevice</name>
  - <argumentList>
    - <argument>
        <name>deviceType</name>
        <direction>in</direction>
        <relatedStateVariable>deviceType</relatedStateVariable>
      </argument>
    - <argument>
        <name>position</name>
        <direction>in</direction>
        <relatedStateVariable>position</relatedStateVariable>
      </argument>
    - <argument>
        <name>requestEntityID</name>
        <direction>in</direction>
        <relatedStateVariable>requestEntityID</relatedStateVariable>
      </argument>
    - <argument>
        <name>outputEntityID</name>
        <direction>out</direction>
        <retval />
        <relatedStateVariable>entityID</relatedStateVariable>
      </argument>
    </argumentList>
  </action>
- <action>
    <name>getSortedDeviceList</name>
  - <argumentList>
    - <argument>
        <name>deviceType</name>
        <direction>in</direction>
        <relatedStateVariable>deviceType</relatedStateVariable>
      </argument>
    - <argument>
        <name>requestEntityID</name>
        <direction>in</direction>
        <relatedStateVariable>requestEntityID</relatedStateVariable>
      </argument>
    - <argument>
        <name>position</name>
        <direction>in</direction>
        <relatedStateVariable>position</relatedStateVariable>
      </argument>
    - <argument>
        <name>deviceList</name>
        <direction>out</direction>
        <retval />
        <relatedStateVariable>deviceList</relatedStateVariable>
      </argument>
    </argumentList>
  </action>
- <action>
    <name>getPosition</name>
  - <argumentList>
    - <argument>
        <name>entityID</name>
        <direction>in</direction>
        <relatedStateVariable>entityID</relatedStateVariable>
      </argument>
    - <argument>
        <name>requestEntityID</name>
        <direction>in</direction>
        <relatedStateVariable>requestEntityID</relatedStateVariable>
      </argument>
    - <argument>
        <name>position</name>
        <direction>out</direction>
        <retval />
        <relatedStateVariable>position</relatedStateVariable>
      </argument>
    </argumentList>
  </action>
```



```
- <action>
    <name>getDeviceLocation</name>
  - <argumentList>
    - <argument>
        <name>entityID</name>
        <direction>in</direction>
        <relatedStateVariable>entityID</relatedStateVariable>
      </argument>
    - <argument>
        <name>requestEntityID</name>
        <direction>in</direction>
        <relatedStateVariable>requestEntityID</relatedStateVariable>
      </argument>
    - <argument>
        <name>location</name>
        <direction>out</direction>
        <retval />
        <relatedStateVariable>location</relatedStateVariable>
      </argument>
    </argumentList>
  </action>
- <action>
    <name>getUserLocation</name>
  - <argumentList>
    - <argument>
        <name>userID</name>
        <direction>in</direction>
        <relatedStateVariable>entityID</relatedStateVariable>
      </argument>
    - <argument>
        <name>requestEntityID</name>
        <direction>in</direction>
        <relatedStateVariable>requestEntityID</relatedStateVariable>
      </argument>
    - <argument>
        <name>location</name>
        <direction>out</direction>
        <retval />
        <relatedStateVariable>location</relatedStateVariable>
      </argument>
    </argumentList>
  </action>
- <action>
    <name>getDeviceListLocation</name>
  - <argumentList>
    - <argument>
        <name>deviceType</name>
        <direction>in</direction>
        <relatedStateVariable>deviceType</relatedStateVariable>
      </argument>
    - <argument>
        <name>location</name>
        <direction>in</direction>
        <relatedStateVariable>location</relatedStateVariable>
      </argument>
    - <argument>
        <name>requestEntityID</name>
        <direction>in</direction>
        <relatedStateVariable>requestEntityID</relatedStateVariable>
      </argument>
    - <argument>
        <name>deviceList</name>
        <direction>out</direction>
        <retval />
        <relatedStateVariable>deviceList</relatedStateVariable>
      </argument>
    </argumentList>
  </action>
```



```
  - <action>
      <name>addLocation</name>
    - <argumentList>
      - <argument>
          <name>locationID</name>
          <direction>in</direction>
          <relatedStateVariable>deviceType</relatedStateVariable>
        </argument>
      - <argument>
          <name>perimeterPoints</name>
          <direction>in</direction>
          <relatedStateVariable>perimeterPoints</relatedStateVariable>
        </argument>
      - <argument>
          <name>output</name>
          <direction>out</direction>
          <retval />
          <relatedStateVariable>output</relatedStateVariable>
        </argument>
      </argumentList>
    </action>
  </actionList>
- <serviceStateTable>
  - <stateVariable sendEvents="no">
      <name>entityID</name>
      <dataType>string</dataType>
    </stateVariable>
  - <stateVariable sendEvents="no">
      <name>userID</name>
      <dataType>string</dataType>
    </stateVariable>
  - <stateVariable sendEvents="no">
      <name>searchString</name>
      <dataType>string</dataType>
    </stateVariable>
  - <stateVariable sendEvents="no">
      <name>distance</name>
      <dataType>string</dataType>
    </stateVariable>
  - <stateVariable sendEvents="no">
      <name>position</name>
      <dataType>string</dataType>
    </stateVariable>
  - <stateVariable sendEvents="no">
      <name>location</name>
      <dataType>string</dataType>
    </stateVariable>
  - <stateVariable sendEvents="no">
      <name>requestEntityID</name>
      <dataType>string</dataType>
    </stateVariable>
  - <stateVariable sendEvents="no">
      <name>deviceType</name>
      <dataType>string</dataType>
    </stateVariable>
  - <stateVariable sendEvents="no">
      <name>deviceList</name>
      <dataType>string</dataType>
    </stateVariable>
  - <stateVariable sendEvents="no">
      <name>output</name>
      <dataType>boolean</dataType>
    </stateVariable>
  </serviceStateTable>
</scpd>
```

Appendix C

Location test data

The following pages contain the test data used for the demonstration of the prototype.

Locations are given in the following format:
Layer: name of the layer (detail_location, collection_location or overview_location)
LocationID: the ID used for the location in the database
Coordinates: (WGS84(hour-min-sec) || WGS84(double) given in the required counterclockwise order
SQL-insert: Gives the SQL statement needed to insert the location into the database
Coordinate inside location: Gives a coordinate that is located within the outline of the location
Coordinate outside location: Gives a coordinate that is located outside the outline of the location

```
--------------------------------------------------------------------------
                             Location Data
--------------------------------------------------------------------------

Layer: detail_location
LocationID: NTNU-Gløshaugen-IT-bygget
Coordinates:
N63-25-2.59550, E10-24-7.14299   || 63.417387678888889, -10.401984163888889
N63-25-3.21176, E10-24-10.12659  || 63.417558822222222, -10.402812941666667
N63-25-3.54724, E10-24-9.75929   || 63.417652011111111, -10.402710913888889
N63-25-3.23860, E10-24-8.03879   || 63.417566277777778, -10.402239972222222
N63-25-3.88359, E10-24-7.38658   || 63.417745441666667, -10.402051827777778
N63-25-3.58595, E10-24-5.93574  || 63.417662763888889, -10.401648816666667
N63-25-2.59550, E10-24-7.14299   || 63.417387678888889, -10.401984163888889

SQL-insert:
INSERT INTO detail_location VALUES('NTNU-Gløshaugen-IT-bygget',
  MDSYS.SDO_GEOMETRY(2003, 8307, NULL,
    MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,1),
    MDSYS.SDO_ORDINATE_ARRAY(63.417387678888889, -10.401984163888889,
      63.417558822222222, -10.402812941666667,
      63.417652011111111, -10.402710913888889,
      63.417566277777778, -10.402239972222222,
      63.417745441666667, -10.402051827777778,
      63.417662763888889, -10.401648816666667,
      63.417387678888889, -10.401984163888889
    )
  )
);

Coordinate inside location:
N63-25-2.99514, E10-24-7.25579   || 63.417498650000000, -10.402015497222222

Coordinate outside location:
N63-25-3.79703, E10-24-8.66584   || 63.417721397222222, -10.402407177777778

--------------------------------------------------------------------------

Layer: collection_layer
LocationID: NTNU-Gløshaugen
N63-24-55.34535, E10-24-15.13605 || 63.415373083333334, -10.404204458333334
N63-24-59.23937, E10-24-29.11624 || 63.416455380555554, -10.408087844444445
N63-25-11.30976, E10-24-13.80953 || 63.419808266666664, -10.403835980555556
N63-25-7.55753,  E10-23-54.30882 || 63.418765980555555, -10.398419116666666

SQL-insert:
INSERT INTO collection_location  VALUES('NTNU-Gløshaugen',
  MDSYS.SDO_GEOMETRY(2003, 8307, NULL,
    MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,1),
    MDSYS.SDO_ORDINATE_ARRAY(63.415373083333334, -10.404204458333334,
      63.416455380555554, -10.408087844444445,
      63.419808266666664, -10.403835980555556,
      63.418765980555555, -10.398419116666666
    )
  )
);

Coordinate inside location:
N63-25-2.99514, E10-24-7.25579   || 63.417498650000000, -10.402015497222222

Coordinate outside location:
N63-25-1.40846, E10-23-45.75714  || 63.417057905555555, -10.396043650000000

--------------------------------------------------------------------------

Layer: overview_layer
LocationID: Trondheim
N63-23-1.43562,  E10-18-48.76641 || 63.383732116666664, -10.313546225000001
N63-23-2.49423,  E10-26-0.39549  || 63.384026175000000, -10.433443191666667
N63-25-50.42746, E10-31-22.92445 || 63.430674294444444, -10.523034569444444
N63-27-19.22504, E10-26-26.81552 || 63.455340288888889, -10.440782088888889
N63-25-55.58342, E10-21-0.80733  || 63.432106505555555, -10.350224258333332
N63-23-1.43562,  E10-18-48.76641 || 63.383732116666664, -10.313546225000001

SQL-insert:
```

```
INSERT INTO collection_location  VALUES('Trondheim',
  MDSYS.SDO_GEOMETRY(2003, 8307, NULL,
    MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,1),
    MDSYS.SDO_ORDINATE_ARRAY(63.383732116666664, -10.313546225000001,
      63.384026175000000, -10.433443191666667,
      63.430674294444444, -10.523034569444444,
      63.455340288888889, -10.440782088888889,
      63.432106505555555, -10.350224258333332,
      63.383732116666664, -10.313546225000001
    )
  )
);

Coordinate inside location:
N63-25-1.40846, E10-23-45.75714  || 63.417057905555555, -10.396043650000000

Coordinate outside location:
N63-24-48.54855, E10-36-2.45588  || 63.413485708333333, -10.600682188888889
```

Label data is given in the following format:
LocationID: The ID of the location that is to be labeled
ContextID: A unique identifier for the context the label is used in
Label: The string that is used as a label for the given location in a given context
SQL-insert: Gives the SQL statement needed to insert the label into the database

```
--------------------------------------------------------------------------
                              Label Data
--------------------------------------------------------------------------
LocationID: NTNU-Gløshaugen-IT-bygget
ContextID: Prosjekt
Label: Kontoret

SQL-insert:
INSERT INTO context_label VALUES(
'NTNU-Gløshaugen-IT-bygget',
'Prosjekt',
'Kontoret'
);

--------------------------------------------------------------------------

LocationID: NTNU-Gløshaugen-IT-bygget
ContextID: Studass
Label: Veilednings-lab

SQL-insert:
INSERT INTO context_label VALUES(
'NTNU-Gløshaugen-IT-bygget',
'Studass',
'Veilednings-lab'
);
```

```
---------------------------------------------------------------------------

LocationID: NTNU-Gløshaugen-IT-bygget
ContextID: Bekjent
Label: Trondheim, Gløshaugen

SQL-insert:
INSERT INTO context_label VALUES(
'NTNU-Gløshaugen-IT-bygget',
'Bekjent',
'Trondheim, Gløshaugen'
);

---------------------------------------------------------------------------

LocationID: NTNU-Gløshaugen
ContextID: Prosjekt
Label: Gløshaugen

SQL-insert:
INSERT INTO context_label VALUES(
'NTNU-Gløshaugen-IT-bygget',
'Prosjekt',
'Trondheim, Gløshaugen'
);

---------------------------------------------------------------------------

LocationID: NTNU-Gløshaugen
ContextID: Bekjent
Label: Trondheim, Gløshaugen

SQL-insert:
INSERT INTO context_label VALUES(
'NTNU-Gløshaugen',
'Bekjent',
'Trondheim, Gløshaugen'
);
```