Location Awareness in UbiCollab

Espen Larsen Segelvik
Morten Larsen Segelvik

Master of Science in Computer Science
Submission date: June 2007
Supervisor: Monica Divitini, IDI
Co-supervisor: Babak Farshchian, IDI
Problem Description

The task is within the context of Ubicollab, a platform for the development of pervasive collaborative applications. The task is focusing on the role of location in collaboration and it aims at designing a solution for location-aware access to resources for collaboration, including both people and artifacts.

Addressing the identified problem will require to work on two core services at the platform level: Collaboration Instances and Collaboration Spaces. The task is expected to result in the design and prototypical development of these components, including support for:
- Basic management of Collaboration Instances
- Management of Collaboration Spaces
- Association of users to Collaboration Spaces based on users' location
- Access to services tailored to users' location and their presence within Collaboration Spaces

The project adopts OSGI and the identified services should be developed as OSGI bundles. The task requires the development, in form of a running prototype, of an application build on Ubicollab and demonstrating the core concepts developed in the thesis.

Supervisor: Monica Divitini, IDI
Abstract

This assignment was given within the UbiCollab project, which is a platform for Ubiquitous Collaboration. Ubiquitous and Pervasive computing were first coined at the beginning of the 90’s, and capture the realization that the computing focus was going to change from the PC to a more distributed, mobile and embedded form of computing. The vision of UbiCollab is to support interaction and collaboration on the internet among geographical dispersed users. UbiCollab treat mobility and ubiquity as inherent properties in social interaction among people. The work in this Master’s thesis will take into account that users are distributed and constantly change location according to their daily routine, which enforces the need for location awareness in UbiCollab. This work will focus upon how location should be represented, managed and used in ubiquitous systems like UbiCollab. To achieve this, we will be working on two related platform components; one for managing locations, and one for managing communities. We start with a problem elaboration, which is followed by an analysis of related research. This leads to platform specific requirements, both functional and non-functional. Based on these requirements we propose a design that meets both the functional and non-functional requirements. The design is followed by an implementation of the design. Finally we show the strengths and weaknesses of the design and implementation by implementing a demonstrator called UbiBuddy. This is an enhanced buddy list built on top of the UbiCollab platform, with specific focus on the collaborative support provided by UbiCollab. In our work we have shown the usefulness of location as a resource in collaboration, both by promoting awareness of users in UbiCollab, and by integrating our work with the component managing services. By doing this integration, we have shown how location can be used as a resource for other platform components aiding collaboration.
Preface

This report documents the work done as a Master’s thesis in computer science at the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU) in Trondheim. The thesis is a contribution to the UbiCollab platform, and is performed in the spring of 2007. The project assignment was given by IDI and Telenor Research & Innovation. The task is focusing on the role of location in collaboration and aims to design a solution for location-aware access to resources for collaboration, including both people and artifacts.

We wish to thank our project supervisor, Professor Dr. Monica Divitini, Dr. Babak A. Farshchian and Otto Helge Nygård for constructive and valuable feedback on the research, the implementation and the writing of the report. We also wish to thank Telenor Research & Innovation for providing us with an office, coffee, fruits, cookies and waffles.

Trondheim, June 17, 2007

Espen Larsen Segelvik

Morten Larsen Segelvik
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>v</td>
</tr>
<tr>
<td>Preface</td>
<td>vii</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Problem domain and general description of the problem</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Task context</td>
<td>2</td>
</tr>
<tr>
<td>1.2.1 Conceptual context</td>
<td>3</td>
</tr>
<tr>
<td>1.2.2 Technical context</td>
<td>4</td>
</tr>
<tr>
<td>1.2.3 Service Oriented Architecture</td>
<td>4</td>
</tr>
<tr>
<td>1.2.4 OSGi</td>
<td>4</td>
</tr>
<tr>
<td>1.3 Motivation, goals and contributions</td>
<td>5</td>
</tr>
<tr>
<td>1.4 Research method</td>
<td>7</td>
</tr>
<tr>
<td>1.5 Report structure</td>
<td>8</td>
</tr>
<tr>
<td><strong>2 Problem Elaboration</strong></td>
<td>11</td>
</tr>
<tr>
<td>2.1 Brief introduction to UbiCollab</td>
<td>12</td>
</tr>
<tr>
<td>2.1.1 Overall UbiCollab architecture</td>
<td>12</td>
</tr>
<tr>
<td>2.1.2 Tools</td>
<td>13</td>
</tr>
<tr>
<td>2.1.3 Applications</td>
<td>13</td>
</tr>
<tr>
<td>2.2 Conceptual Overview</td>
<td>14</td>
</tr>
<tr>
<td>2.2.1 Collaboration Space</td>
<td>14</td>
</tr>
<tr>
<td>2.2.2 Collaboration Instance</td>
<td>14</td>
</tr>
<tr>
<td>2.2.3 Relation between Collaboration Spaces and Collaboration Instances</td>
<td>14</td>
</tr>
<tr>
<td>2.3 Our focus</td>
<td>15</td>
</tr>
<tr>
<td>2.4 Scenario</td>
<td>16</td>
</tr>
<tr>
<td>2.4.1 UbiBuddy</td>
<td>17</td>
</tr>
<tr>
<td>2.4.2 Analysis of scenarios to identify high level needs</td>
<td>17</td>
</tr>
<tr>
<td>2.5 Subgoals</td>
<td>18</td>
</tr>
<tr>
<td><strong>3 Related Research</strong></td>
<td>21</td>
</tr>
<tr>
<td>3.1 Location-Aware Computing</td>
<td>22</td>
</tr>
<tr>
<td>3.2 Location Technologies</td>
<td>24</td>
</tr>
<tr>
<td>3.3 Location Models</td>
<td>25</td>
</tr>
<tr>
<td>3.3.1 Physical location models</td>
<td>25</td>
</tr>
<tr>
<td>3.3.2 Geographical location models</td>
<td>26</td>
</tr>
<tr>
<td>3.3.3 Hybrid location models</td>
<td>26</td>
</tr>
<tr>
<td>3.3.4 Location Data Storage</td>
<td>27</td>
</tr>
</tbody>
</table>
3.4 Place Lab .............................................................. 27
3.5 Summary .............................................................. 28

4 Platform Requirements ............................................... 31
4.1 Analysis of UbiBuddy ................................................. 32
4.2 Platform requirements ............................................... 33
4.3 High level use cases ................................................ 33
  4.3.1 Collaboration Space ............................................ 34
  4.3.2 Collaboration Instance ........................................ 38
4.4 Platform specific requirements .................................... 42
  4.4.1 Functional requirements ...................................... 42
  4.4.2 Non-functional requirements .................................. 45

5 Design .................................................................. 47
5.1 Background ........................................................... 48
5.2 UbiCollab architecture ............................................. 48
  5.2.1 Tools ............................................................... 49
  5.2.2 Applications ....................................................... 51
  5.2.3 UbiHome ........................................................... 51
5.3 Collaboration Instance Manager .................................. 51
  5.3.1 Sequence Charts ................................................ 51
  5.3.2 API ................................................................. 54
5.4 Collaboration Space Manager ....................................... 55
  5.4.1 Definition of a Collaboration Space ......................... 56
  5.4.2 Location Model ................................................... 56
  5.4.3 Personal vs. shared Collaboration Spaces .................... 56
  5.4.4 Mapping from coordinates to Collaboration Spaces ......... 58
  5.4.5 Granularity issues .............................................. 59
  5.4.6 Sequence Charts ............................................... 59
  5.4.7 API ................................................................. 63
5.5 Putting it all together ............................................... 64

6 Implementation .......................................................... 67
6.1 Implementation overview ........................................... 68
6.2 Collaboration Space Manager ...................................... 69
  6.2.1 API ............................................................... 69
  6.2.2 Class diagram ..................................................... 70
6.3 csmTool ................................................................. 70
6.4 Collaboration Instance Manager .................................... 71
  6.4.1 API ............................................................... 71
  6.4.2 Class diagram ..................................................... 73
6.5 cimTool ................................................................. 73
6.6 CIUpdaterTool ......................................................... 73
  6.6.1 Class diagram ..................................................... 75
6.7 UbiHome ................................................................. 75
6.8 Implementation vs. Design ........................................... 76
  6.8.1 Collaboration Space Manager ................................. 76
  6.8.2 Collaboration Instance Manager ............................... 77

7 Demonstrator ............................................................ 79
7.1 Application - UbiBuddy ............................................. 80


## List of Tables

<table>
<thead>
<tr>
<th>3.1</th>
<th>P3 system framework techniques summary</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2</td>
<td>Pros and cons of physical location models</td>
<td>26</td>
</tr>
<tr>
<td>3.3</td>
<td>Pros and cons of geographical location models</td>
<td>26</td>
</tr>
<tr>
<td>3.4</td>
<td>Pros and cons of hybrid location models</td>
<td>27</td>
</tr>
<tr>
<td>4.1</td>
<td>Scenario actions at the platform level</td>
<td>32</td>
</tr>
<tr>
<td>4.2</td>
<td>High Level Needs And High Level Requirements</td>
<td>33</td>
</tr>
<tr>
<td>4.3</td>
<td>Creation of Collaboration Space</td>
<td>35</td>
</tr>
<tr>
<td>4.4</td>
<td>Deletion of Collaboration Space</td>
<td>35</td>
</tr>
<tr>
<td>4.5</td>
<td>Modification of Collaboration Space</td>
<td>36</td>
</tr>
<tr>
<td>4.6</td>
<td>Get current Collaboration Space</td>
<td>36</td>
</tr>
<tr>
<td>4.7</td>
<td>Automatic update of current Collaboration Space</td>
<td>37</td>
</tr>
<tr>
<td>4.8</td>
<td>Manual update of current Collaboration Space</td>
<td>38</td>
</tr>
<tr>
<td>4.9</td>
<td>Creation of Collaboration Instance</td>
<td>38</td>
</tr>
<tr>
<td>4.10</td>
<td>Deletion of Collaboration Instance</td>
<td>40</td>
</tr>
<tr>
<td>4.11</td>
<td>Association of users to Collaboration Instances</td>
<td>40</td>
</tr>
<tr>
<td>4.12</td>
<td>Modification of Collaboration Instance</td>
<td>41</td>
</tr>
<tr>
<td>4.13</td>
<td>Sharing of data in a Collaboration Instance</td>
<td>41</td>
</tr>
<tr>
<td>4.14</td>
<td>Get Collaboration Instance data</td>
<td>42</td>
</tr>
<tr>
<td>4.15</td>
<td>Functional requirements</td>
<td>43</td>
</tr>
<tr>
<td>4.16</td>
<td>Non-functional requirements</td>
<td>45</td>
</tr>
<tr>
<td>5.1</td>
<td>Description of the different components in UbiCollab</td>
<td>49</td>
</tr>
<tr>
<td>5.2</td>
<td>Description of Collaboration Space properties</td>
<td>56</td>
</tr>
<tr>
<td>6.1</td>
<td>Design API vs. Implementation API - Collaboration Space Manager</td>
<td>77</td>
</tr>
<tr>
<td>6.2</td>
<td>Design API vs. Implementation API - Collaboration Instance Manager</td>
<td>78</td>
</tr>
<tr>
<td>7.1</td>
<td>Actions at the platform level</td>
<td>86</td>
</tr>
<tr>
<td>7.2</td>
<td>Fulfillment of functional requirements</td>
<td>87</td>
</tr>
<tr>
<td>7.3</td>
<td>Fulfillment of non-functional requirements</td>
<td>88</td>
</tr>
<tr>
<td>A.1</td>
<td>Implementation Attachments</td>
<td>iii</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Basic SOA architecture</td>
<td>4</td>
</tr>
<tr>
<td>1.2</td>
<td>Research method</td>
<td>7</td>
</tr>
<tr>
<td>2.1</td>
<td>The different components in UbiCollab</td>
<td>13</td>
</tr>
<tr>
<td>2.2</td>
<td>Collaboration Spaces</td>
<td>15</td>
</tr>
<tr>
<td>2.3</td>
<td>Location Layers</td>
<td>16</td>
</tr>
<tr>
<td>4.1</td>
<td>Use Case: Collaboration Space</td>
<td>34</td>
</tr>
<tr>
<td>4.2</td>
<td>Use Case: Collaboration Instance</td>
<td>39</td>
</tr>
<tr>
<td>5.1</td>
<td>Architectural overview</td>
<td>50</td>
</tr>
<tr>
<td>5.2</td>
<td>Create Collaboration Instance Sequence</td>
<td>52</td>
</tr>
<tr>
<td>5.3</td>
<td>Delete Collaboration Instance Sequence</td>
<td>52</td>
</tr>
<tr>
<td>5.4</td>
<td>Add user to Collaboration Instance Sequence</td>
<td>53</td>
</tr>
<tr>
<td>5.5</td>
<td>Delete user from Collaboration Instance Sequence</td>
<td>53</td>
</tr>
<tr>
<td>5.6</td>
<td>Modify Collaboration Instance Sequence</td>
<td>53</td>
</tr>
<tr>
<td>5.7</td>
<td>Publish data in a Collaboration Instance Sequence</td>
<td>54</td>
</tr>
<tr>
<td>5.8</td>
<td>Get published Collaboration Instance data Sequence</td>
<td>54</td>
</tr>
<tr>
<td>5.9</td>
<td>Collaboration Space Manager API</td>
<td>55</td>
</tr>
<tr>
<td>5.10</td>
<td>Mapping between physical locations and Collaboration Spaces</td>
<td>58</td>
</tr>
<tr>
<td>5.11</td>
<td>Collaboration Space Hierarchy</td>
<td>59</td>
</tr>
<tr>
<td>5.12</td>
<td>Add Collaboration Space Sequence</td>
<td>60</td>
</tr>
<tr>
<td>5.13</td>
<td>Delete Collaboration Space Sequence</td>
<td>60</td>
</tr>
<tr>
<td>5.14</td>
<td>Modify Collaboration Space Sequence</td>
<td>61</td>
</tr>
<tr>
<td>5.15</td>
<td>Get Current Collaboration Space Sequence</td>
<td>61</td>
</tr>
<tr>
<td>5.16</td>
<td>Get Nearest Collaboration Space Sequence</td>
<td>62</td>
</tr>
<tr>
<td>5.17</td>
<td>Set Current Collaboration Space Sequence</td>
<td>62</td>
</tr>
<tr>
<td>5.18</td>
<td>Automatic Update of Collaboration Space Sequence</td>
<td>63</td>
</tr>
<tr>
<td>5.19</td>
<td>Collaboration Space Manager API</td>
<td>63</td>
</tr>
<tr>
<td>5.20</td>
<td>Login Sequence</td>
<td>65</td>
</tr>
<tr>
<td>6.1</td>
<td>Implementation overview</td>
<td>68</td>
</tr>
<tr>
<td>6.2</td>
<td>Collaboration Space Manager API</td>
<td>70</td>
</tr>
<tr>
<td>6.3</td>
<td>Collaboration Space Manager Class Diagram</td>
<td>71</td>
</tr>
<tr>
<td>6.4</td>
<td>csmTool</td>
<td>72</td>
</tr>
<tr>
<td>6.5</td>
<td>Collaboration Instance Manager API</td>
<td>73</td>
</tr>
<tr>
<td>6.6</td>
<td>Collaboration Instance Manager Class Diagram</td>
<td>74</td>
</tr>
<tr>
<td>6.7</td>
<td>cimTool</td>
<td>74</td>
</tr>
<tr>
<td>6.8</td>
<td>Collaboration Instance Updater Tool Class Diagram</td>
<td>75</td>
</tr>
</tbody>
</table>
Listings

A.1 Create statements for the UbiHome MySQL database . . . . . . . iii
Chapter 1

Introduction

*The best way to predict the future is to invent it*

-Alan Kay
1.1 Problem domain and general description of the problem

The problem domain of this task is within the category of location awareness, which aims to support location awareness for both users and applications. Location awareness is a sub domain of context awareness, where the context variable used is location. According to [9] "awareness is an understanding of the activities of others, which provides a context for your own activity." Translated to location this definition becomes: "location awareness is the understanding of the others' position in the spatial environment" [20]. Location is an important attribute in order to promote social awareness, which aims to provide awareness of other users. This is increasingly important in collaborative settings. UbiCollab is a platform that aims to support collaboration in ubiquitous environments where users are distributed. In collaborative settings location can be used in a various ways. Applications can be location-aware, i.e. the applications can adapt and change based on users location. At home you can access different embedded services, while at work the services at home become unavailable. A second usage of location is by the users themselves. If the location of a user is shared among the collaborating participants, they easily become aware of each others presence. For distributed users, this becomes even more important. For instance, if a work colleague knows that you are home, he or she might not disturb you with work related issues. However knowing that you are in the canteen he or she might send you a message. As a consequence, location awareness is an important resource for collaboration, which can be utilized in various ways. When designing location aware systems, it is important to keep in mind that the location represents different meanings to different communities. As stated in [10], different people associate completely different representation of the same geographical location:

Brian is at coordinates (X,Y). For his colleagues this means he is out of office, for his child it means "Daddy is here, fortunately today I am not the last one leaving the kindergarten", for his friends organizing a surprise birthday party, it means he is on his way home.

As a consequence when designing location-aware systems, one should take this property into account. Apart from sharing information about a user’s location this should be utilized when developing location-aware applications. At home you might want to share certain services to your family and friends, and at work, you only want to share “work related services” to your colleagues. From this we can see that taking into account the different communities a user operate in, each community need customized data based on the location. Seeing these challenges in this research area, we will focus our work on location awareness in UbiCollab.

1.2 Task context

The task will focus upon location awareness in distributed collaborative settings. UbiCollab is a platform for collaborative applications, supporting collaboration on the internet. What we wish to achieve is to introduce a new level of location and community awareness into UbiCollab. Making UbiCollab location-aware means that the platform has a component, which provides the location as a resource for collabo-
ration. By introducing this component as a standalone service at the platform level, each application and component developed on the platform, can utilize this concept in various ways to aid collaboration. This component itself is not enough to provide the wanted functionality. Due to how location affects communities in different ways, working with the location part also involves the management of communities. Following is the conceptual and the technical context of our task.

1.2.1 Conceptual context

With ubiquitous computing one often make use of embedded services in order to aid the user to perform different tasks. In distributed collaborative settings tasks must be coordinated among dispersed individuals. Awareness of each other as well as sharing of information and artifacts embedded in each participant’s environment helps the coordination and execution of these. In order to achieve this, location awareness in UbiCollab is required. To provide an acceptable level of location awareness, one should take into account the viewpoint of communities. As stated in the previous section, different users associate locations differently, thus our work should take this into account. I.e. we will be working on two levels in UbiCollab; the location level and the community level. We will now elaborate about location and community.

Location

People normally do not reason in terms of geographical coordinates, but relates to places. I.e. higher level representations to which they associate meanings[10]; we are at home, at work, at the theater, etc. Therefore it is important to introduce location in a human understandable way. Presenting location information in a human understandable way is not enough. Computer reasoning in human representations of locations are tricky, and requires a lower level of abstraction. Even though the high level is understandable for humans, computers need measurable properties like GPS coordinates. Location in UbiCollab are captured in Collaboration Spaces. A Collaboration Space captures the characteristics of physical location in different levels of abstractions. It is important to provide a low level abstraction in form of coordinates in order to provide applications the possibility to do different calculation and adaptations based on location. The higher abstractions levels are important in order to enable human reasoning about locations. Due to the fact that a Collaboration Space captures physical characteristics of places where collaboration takes place, each Collaboration Space have different properties. These can for instance be shape and services available in that location. Capturing the connection between services and Collaboration Spaces enables automatic adaptation to locations in UbiCollab.

Community

The term community can be defined in various ways. At this stage of the UbiCollab project it is hard to foresee the user group, thus it is important to have a general description that can be utilized in as many communities as possible. We feel that the following definition of community is appropriate for UbiCollab:

Espen Larsen Segelvik and Morten Larsen Segelvik
1. Introduction

A number of individuals, defined by formal or informal criteria of membership, who share a feeling of unity or are bound together in relatively stable patterns of interaction[27]. Communities in UbiCollab are realized as Collaboration Instances. If a user wants to share information to other users, this is done in a Collaboration Instance. This enables other members of the community to access information shared by other users.

1.2.2 Technical context

The task addresses different research problems within the UbiCollab project. UbiCollab provides a platform that captures the commonalty of collaborative applications and provides generic mechanisms for applications to be built without extensive coding. UbiCollab is therefore following an open innovation approach where third party applications play an equally central role as the platform itself. Integration with physical environment where collaboration happens is a key aspect of UbiCollab[11]. One of the main architectural ideas of UbiCollab is a service oriented architecture (SOA), implemented in OSGi. This is essentially the same restriction given from ASTRA in the autumn project[26]. In the next sections we give a brief introduction to SOA and OSGi, which is the same overview as we wrote in the autumn project.

1.2.3 Service Oriented Architecture

A service-oriented architecture (SOA) is essentially a collection of services, which communicate with each other. The communication can vary from simple data passing to involving two or more services coordinating some activity. SOA is an evolution of the Component Based Architecture, Interface Based Design (Object Oriented) and Distributed Systems of the 1990s [19]. SOA is a way of reorganizing software applications and infrastructure into a set of interacting services[21]. The basic architecture is shown in Figure 1.1.

![Figure 1.1: Basic SOA architecture][1]

1.2.4 OSGi

The OSGi Alliance was founded in 1999 in order to create an open specification for networked delivery of managed services to local networks and devices. The OSGi Service Platform Core Specification[3] "delivers an open, common architecture for service providers, developers, software vendors, gateway operators and equipment vendors to develop, deploy and manage services in a coordinated fashion.” The

---

[1]: #/figure1.1.png
OSGi Framework forms the core of the OSGi Service Platform Specifications, which provide a general-purpose, secure and managed Java framework for deploying applications. These applications are known as bundles. The bundles are created as a normal Java application with the exception of replacing the main-method with an activator class which implements the OSGi interface BundleActivator. Bundles are deployed as a Java ARchive (JAR) file. These files are used to store applications and their resources. A bundle JAR file contains the resources necessary to provide functionality and a manifest file describing the contents of the JAR file and providing information about the bundle. In addition the JAR file can contain optional documentation in a directory called OSGI-OPT. When a bundle is started, the functionality of the bundle is provided to other bundles installed in the OSGi Service Platform. The OSGi framework manages installation and update of bundles in a dynamic and scalable fashion. To achieve this, the framework manages dependencies between bundles and services in detail.

The functionality of the OSGi framework is divided into five different layers. These layers are dealing with different aspects of the framework. We will here give a brief overview of these layers:

Security Layer: This is an optional layer based on the Java 2 security architecture. It provides the infrastructure needed in order to deploy and manage application that must run in a controlled environment.

Module Layer: This layer defines a modularization model for java. This layer has strict rules for sharing Java packages between bundles or hiding packages from other bundles.

Life Cycle Layer: The Life Cycle Layer provide an API to bundles in order to provide a runtime model for bundles. It defines how bundles are started, stopped, installed, updated and uninstalled. This layer requires the Module Layer.

Service Layer: This Layer is provided to bundle-developers in order to provide a dynamic, concise and consistent programming model, by de-coupling a bundle’s implementation from its interface.

Actual Services: The top layer consisting of all services running on top of the other layers. An OSGi service is implemented as a service object owned by and run within a bundle. The service is semantically defined by its service interface.

1.3 Motivation, goals and contributions

We will now present the motivation for our work, goals and contributions.

Motivation

The motivation for our work is to introduce location awareness to UbiCollab. The introduction of location awareness to UbiCollab will bring a range of new possibilities to applications built on top of it. One can customize, configure and share
information based on location, tailored to each community, thus aiding collaboration. This introduction will also help users of UbiCollab gain social awareness about other users, easing the execution of different tasks related to collaboration.

Goals

The main goal of this thesis is to design, implement, and evaluate UbiCollab components to support the usage of location information as a resource for collaboration in distributed and ubiquitous settings. The platform will be implemented with SOA principles, meaning that we will be working on two different component related to each other; one for managing locations, and one for managing communities. Location is the main focus of our work, and throughout this thesis we will therefore focus mostly on the location part of the platform.

Contributions

The report will present the following set of contributions made by our work within UbiCollab:

- **C1**: Problem elaboration
  - **C1.1**: Presentation of scenario
  - **C1.2**: Analysis of the scenario
- **C2**: Platform requirements
- **C3**: Design
  - **C3.1**: Design of a component managing Collaboration Spaces
  - **C3.2**: Design of a component managing Collaboration Instances
- **C4**: Implementation
  - **C4.1**: Implementation of a component managing Collaboration Spaces
  - **C4.2**: Implementation of a component managing Collaboration Instances
  - **C4.3**: Integrate our work with the component managing services
- **C5**: Demonstration of the implementation
- **C6**: Evaluation of the design and implementation

The first contribution C1 regards the problem elaboration. Within this contribution we have a presentation of a relevant scenario for our work, followed by an analysis to identify high level requirements. This will lead to the contribution C2; Platform requirements. This includes both functional and non-functional requirements for the components managing Collaboration Spaces and the Collaboration Instances. From the platform requirements we will propose a design of the component managing Collaboration Spaces, and a simple design of the component managing Collaboration Instances. This is contribution C3. From the design we will implement a prototype of these components, and integrate our implementation with the component managing services. This is contribution C4. Contribution C5 is the demonstration of the implementation. The implementation will be demonstrated with an application
1.4 Research method

The research method for this thesis is illustrated in Figure 1.2. The figure also shows where the contributions defined in the previous section are realized. We start with the problem elaboration, where we illustrate the problem with a scenario. The scenario is then analyzed in order to identify high level needs. With these requirements as a starting point, we look into related research and identify platform requirements. These platform requirements lead to the creation of a design, followed by an implementation. To show the functionality and evaluate the design and implementation, we will create a demonstrator. The demonstrator will be a location-aware application running on top of UbiCollab, supporting customization of services based on location and providing awareness of users.

Figure 1.2: Research method
1.5 Report structure

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

2 Problem Elaboration: This chapter elaborates the problem, starting with an introduction to UbiCollab. We then elaborate different concepts related to our work. From these concepts we define our focus within location awareness in UbiCollab. Then we further elaborate our task with a scenario illustrating the basic functionalities required for our design and implementation. This design is analyzed in order to identify high level needs. Finally, we define research subgoals that this thesis wishes to achieve.

3 Related Research: In this chapter we will look into research related to our task. First we look into related research with regards to location awareness within collaborative systems. This is followed by a brief overview of different position technologies that can be used when introducing location awareness in UbiCollab. Further we present different location models used in location-aware systems. Finally we look into a system called Place Lab, which addresses how locations can be managed with the help of user interaction.

4 Platform Requirements: This chapter starts with an analysis of the application presented in the scenario in Chapter 2. Based on the analysis we deduce platform specific requirements. From these requirements we deduce use cases showing the usage of UbiCollab. From these use cases we deduce functional and non-functional requirements for our contribution to UbiCollab.

5 Design: In this chapter we will propose a design of the components responsible for the management of communities and location. We start with a presentation of the basic architecture of UbiCollab, and define our work within the different UbiCollab components. We start with a basic design of the Collaboration Instance Manager. This is followed by a thorough design of the Collaboration Space Manager, which is our main contribution to UbiCollab.

6 Implementation: This chapter gives a thorough description of the platform components implemented during this thesis. We start with the Collaboration Space Manager, which is followed by the Collaboration Instance Manager. Finally we analyze how the implemented API conforms to the designed API.

7 Demonstrator: In this chapter we describe a demonstrator implemented to demonstrate and evaluate our work. We illustrate the functionality of the demonstrator with a scenario combined with screenshots. We then use the demonstrator to evaluate our design with regards to functional and non-functional requirements, and identify limitation with the implementation.

8 Conclusion: In this chapter we start with a summary of our thesis during the spring of 2007. This is followed by a discussion of our contributions to UbiCollab. Finally we identify further work that should be considered in future development of UbiCollab.
1.5. Report structure

Appendices:

• *Appendix A* Installation instructions of our implementation.
• *Appendix B* UbiCollab Architecture White Paper.
Chapter 2

Problem Elaboration

*If we can really understand the problem, the answer will come out of it, because the answer is not separate from the problem.*

-Jiddu Krishnamurti
In this chapter we will elaborate the problem domain, starting with a brief introduction to UbiCollab. This is followed by an elaboration of concepts related to UbiCollab. Then we describe of our focus within UbiCollab and location aware collaborative computing. Then we will illustrate the concepts with a scenario, followed by an analysis of the scenario in order to identify high level needs. Finally, this will lead to subgoals this thesis wishes to achieve.

2.1 Brief introduction to UbiCollab

In this section we will give a brief introduction to the basic components of this platform, starting with the overall UbiCollab architecture. This is followed by the definition of two concepts in UbiCollab; Tools and Applications.

2.1.1 Overall UbiCollab architecture

UbiCollab is early in the design phase and at the current state, the architecture consists of the components shown in Figure 2.1. In the list below we give a brief description of these components. We will only be working on the Collaboration Space Manager and the Collaboration Instance Manager, which are colored blue in the figure. The rest of the components will not be further elaborated in this report, due to the fact that they fall out of the scope of our task. The platform components are in UbiCollab realized as Managers. For instance, the component responsible for the Collaboration Instances, are realized as the Collaboration Instance Manager.

- **Collaboration Instance Manager**: Responsible for managing a user’s Collaboration Instances and the data shared in each Collaboration Instance. Collaboration Instance Manager is also responsible for enabling the sharing of Collaboration Instance object among Collaboration Instance members distributed across a network.

- **Collaboration Space Manager**: Responsible for creating Collaboration Spaces, managing existing Collaboration Spaces, querying Collaboration Spaces based on point coordinates and locate points in Collaboration Spaces.

- **Service Discovery Manager**: The Service Discovery Manager responsible for finding services in the user’s surrounding. It gets a service query and returns a service address referring to a service.

- **Service Domain Manager**: Responsible for installing services, managing data related to services, and answering queries about availability of services.

- **Service Ontology Manager**: Responsible for maintaining ontology data, and providing search and query mechanisms for UbiCollab service ontology.

- **Service Composer**: Responsible for maintaining/validating/managing service compositions, for executing service compositions, and for sharing of service compositions.

- **Identity Manager**: Responsible for managing (creating, defining, deleting) virtual identities for a user.
2.1.2 Tools

Tools are components that ties different parts of the UbiCollab platform together. One tool can for instance be used to share the Collaboration Space of a user in a Collaboration Instance. This enables other users to see the current Collaboration Space of the user. The same tool could be extended to activate services connected to the Collaboration Space based on location. Tools can be customized to use information about the different platform components in order to tailor the platform for a particular purpose. Tools are also responsible for altering component specific information. I.e. a tool for the Collaboration Space Manager is responsible for creating, deleting and modifying Collaboration Spaces.

2.1.3 Applications

Applications are built on top of the platform and use the API provided by the Collaboration Instance Manager. An application in the context of UbiCollab can
be seen as a view on the different Collaboration Instances. Applications can use tools provided to alter data in the platform components in order to perform different tasks.

### 2.2 Conceptual Overview

We will in this section discuss more thoroughly the concepts of Collaboration Spaces, Collaboration Instances and the relation between these two terms.

#### 2.2.1 Collaboration Space

The Collaboration Space is a representation of physical locations, which are illustrated in Figure 2.2. The physical locations Monica’s home, Lola’s home and the university are represented as Collaboration Spaces. As the figure illustrates, people associate different meanings to the same physical location. Monica is a professor at the university, and she associates this location as her "office". Lola is a student at the same university, and here is the same location is associated to "school". This shows how the same location as associated differently among individuals. The same location can also be associated differently by the same individual. For instance, Monica is working at home. Then this location is associated as her "home office". When she is not working at home, the place is associated as her "home".

In the figure there are services connected to Collaboration Spaces. This is done in order to show how location can be used as a resource in different locations. When Monica is working at home, she has a printer service available which she needs for printing work related documents. When she is not working at home she has a light control service in order to adjust the light according to her preferences. When she is in her office at the University, she has another printer available. This shows how UbiCollab should adapt according to the user’s location and social setting. These are all important aspects that need to be taken into consideration when designing the Collaboration Space Manager.

#### 2.2.2 Collaboration Instance

The Collaboration Instance is a representation of communities in UbiCollab. In order to provide this functionality, the Collaboration Instance should contain information about the members of the community, and other data needed in collaborative settings. The users of a Collaboration Instance have the possibility to share different information to the other members, for instance their availability for collaboration, location and services.

#### 2.2.3 Relation between Collaboration Spaces and Collaboration Instances

There are relations between Collaboration Spaces and Collaboration Instances we need to consider. When a user enters a Collaboration Space, different properties influences the collaboration among participants in Collaboration Instances. A user
might have a service shared within a Collaboration Instance, enabling other users to display information through a physical device. However, if the user moves to another Collaboration Space, the service displaying information should become unavailable. There are also other properties related to Collaboration Spaces that affect the ability to collaborate. For instance a user’s mental state affects the ability to collaborate. When a user is at home, he or she might not want to be disturbed by work colleagues. This enforces the need of changing system behavior among Collaboration Instances with regards to the Collaboration Space the user reside in. When sharing Collaboration Space information in a Collaboration Instance, other users become aware of the location of the different participants.

2.3 Our focus

Our main focus in UbiCollab is the management of location models. This is a crucial element in order to provide location-aware computing. Figure 2.3 shows four layers in collaborative location-aware computing, which needs to be considered when introducing location awareness in UbiCollab. The bottom layer, Location Technologies is typical GPS, GSM positioning and wireless location technologies providing coordinates to the location model layer. The location technologies are
not the focus of our task. We need coordinates to the location model, but the origin of these is irrelevant.

The Location Model layer is the representation of locations in the system. There are many different location models which will be elaborated in Chapter 3. However, this will not be the main focus in our thesis. We will focus upon the management and distribution of the location model. This is shown in the second layer, Location Model Management.

The top layer in the figure, Collaborative Location-Aware collaboration uses all the three lower levels in order to aid collaboration. The location is an important attribute in collaboration; it gives much awareness information about the users and can be utilized in order to retrieve services and devices.

![Figure 2.3: Location Layers](image_url)

### 2.4 Scenario

This section gives a description of a scenario using an application named UbiBuddy. This is a ubiquitous buddy list used in different collaborative settings. It is location-aware, and adapts according to predefined settings and location. UbiBuddy relates to our work because it is built on top of the UbiCollab platform, and utilizes the functionality provided. The scenario illustrates how location can be used as a resource for collaboration. The scenario followed by an analysis, in order to identify high level needs.
2.4. Scenario

2.4.1 UbiBuddy

**Action 1:** Pedro is using UbiBuddy to stay in touch with his family and friends. UbiBuddy is on his Tablet PC which he usually brings wherever he goes.

**Action 2:** He currently has set up two communities, friends and family. He has also defined a Collaboration Space connected to his office.

**Action 3:** He has just met a group of environmental activists and he decides to use UbiBuddy to stay in touch with them, so he creates a new community and he adds Jorge and Lola. He will ask later to the others to register to UbiCollab, so that he can add them.

**Action 4:** Pedro can now see the new community in his list of communities with the registered participants.

**Action 5:** He decides to set his status as not available to the new community when he is at school.

**Action 6:** Pedro has just moved into his new apartment and he wants to configure the devices that he has available so that he can use them easily to stay in touch.

**Action 7:** He has just bought a lamp and he decides to put it in the corridor and to make it available to his friends as an awareness display.

**Action 8:** Birgit, one of Pedro’s friends, connects to UbiBuddy and she sees that Pedro is at home and he has made available the cool lamp she has suggested him to buy. She clicks on the lamp icon twice to send him a quick message. Maybe he will get curious and check who is on-line.

**Action 9:** Pedro actually gets curious of who is turning on his lamp, so he checks who is on-line by looking at his tablet. He sees that Birgit is on-line and he calls her on the VOIP tool that she has made available.

**Action 10:** It is now time to go to work. When Pedro arrives at his office Ubibuddy adapts to his new location. The main screen in his office becomes now the main display for visualizing awareness about his contacts.

**Action 11:** Pedro is in his office. When there he has made available his coffee machine as a service to his friends. They like that they can use UbiBuddy to check when he is in his office and that they can turn on the coffee machine before popping by for a break.

**Action 12:** Pedro is traveling to Spain and he has no equipment with him. He borrows a mobile phone from his cousin. The phone has the UbiCollab platform installed and he can easily get all his settings.

2.4.2 Analysis of scenarios to identify high level needs

The scenario above describes several different high level needs that should be considered when specifying the platform requirements. In this analysis we will focus on location and community aspects of this scenario.

- Management of communities
The management of communities is described in action 2-4. Pedro already has UbiCollab installed in order to stay in touch with his family and friends (two communities). In action 3 he meets a group of environmental activist, and decides to create a new community for this purpose. Based on these actions, we can see the need for management of communities in UbiCollab.

- Management of locations used for collaboration

The management of locations is described in action 2, 6 and 7. In action 2 he already has a Collaboration Space defined, that UbiCollab adapts to when enters this. In action 6 he has just moved into his new apartment, and he decides to put a lamp in the corridor available as an awareness display. If he already has the home registered in his location model, he can just connect the lamp to this Collaboration Space. However, if he has his old apartment registered, he can either modify this Collaboration Space, or add a new one. In order to adapt UbiCollab to his new home, it is necessary to provide the possibility to create, delete and modify locations.

- Provide location as a resource to other components in UbiCollab

This requirement arises from action 8 and 11. In these actions Pedro use locations as a resource to automatically customize and share devices with participants in his communities. Different services becomes available, depending upon Pedro’s location; at home he has a lamp available, and at the office a coffee machine is available. In order to achieve this, location should be provided as a resource for the components managing devices in a way that minimizes the user interaction required.

- UbiCollab should adapt to locations

In action 5 he has configured his status as not available for the new community when he is at school. This could be done either manually, requiring user interaction, or automatically. Last semester we designed and implemented a location-aware availability manager in ASTRA. This could be integrated with the UbiCollab platform, taking advantage of the location management. Further work on this component will not be done in this thesis.

- Users can share information to communities

In order to perform tasks in collaborative distributed settings, it is important to be able to share information among different communities. This could be awareness information, such as current location. In action 11 he makes his coffee machine available as a service to his friends when he is in his office. When the coffee machine is available for his friends, they know that he is available for a quick meeting. If the coffee machine starts brewing coffee, he knows that one or more of his friends are popping by. Based on this action we see the need for sharing information among members of a community.

2.5 Subgoals

Based on the problem elaboration we have decided upon the following subgoals this thesis wishes to address:

- **Subgoal 1:** Identification of requirements at the platform level to make location information available as a resource for collaboration in UbiCollab.
2.5. Subgoals

- **Subgoal 2**: Define and implement a location model that fulfills UbiCollab requirements and the need to associate different descriptions to places.

- **Subgoal 3**: Design and prototype mechanisms for supporting the sharing of location information within communities.

- **Subgoal 4**: Design and prototype mechanisms to use location information to support the service management.

- **Subgoal 5**: Implement a demonstrator showing the strengths and weaknesses of our design and implementation.

Subgoal 1 is needed in order to create a suitable design of the components related to location on the platform. Designing these components without a requirement specification, would probably lead to a poor design, thus this subgoal is important. From subgoal 1 we will move forward to subgoal 2. This subgoal will be the most important to fulfill, due to the fact that other components rely on information retrieved from this component. Using the results from subgoal 2, we will show the diversity of location awareness in other parts of the platform. This is done by fulfilling the subgoals 3 and 4. Finally we will evaluate the results in subgoal 5, by implementing a demonstrator showing the use of the UbiCollab platform and the fulfillment of the main goal:

To design, implement, and evaluate UbiCollab components to support the usage of location information as a resource for collaboration in distributed and ubiquitous settings.
Chapter 3

Related Research

Research is to see what everybody else has seen, and to think what nobody else has thought.
-Albert Szent-Gyorgyi
In this chapter we will start with an overview of location-aware computing. This is followed by an overview of some technologies available for retrieving position. This will give us an overview of different hardware aspects that needs to be considered when designing and implementing location awareness in UbiCollab. Then, we will look into different location models, for storing location information. Taking this into consideration in our design is very important due to the functionality the different models provide. Then we will give a quick introduction to different strategies for storing these models. Finally, we will give an overview of Place Lab.

### 3.1 Location-Aware Computing

The evolution of mobile computing, location sensing, and wireless networking has created a new class of computing: location-aware computing. Mobile computing is often associated with small devices such as PDAs with wireless network access. Location-aware computing systems respond to a user’s location, either spontaneously or when activated by a user request. There are many ways of retrieving the location of a user. In Section [3.2](#) a brief overview of common technologies is given. Due to the variety of dimension location sensing technology can vary; no single location-sensing technology is likely to become dominant. For instance there are different technologies for sensing location indoors and outdoors. This has major impact on the infrastructure needed in location aware computing systems. Taking this into consideration, it is important that systems developed have a technology independent platform, freeing application writers from the specifics of location-sensing technologies. [22](#) identifies four main requirements for the API provided by the platform regarding location awareness:

- Be based on an open rather than proprietary standard
- Mask technology-dependent attributes of the underlying technology
- Allow specification of desired accuracy and discovery of actual accuracy
- Be capable of dynamically combine location information from multiple sources in a manner that is transparent to applications.

When it comes to implementing location-aware systems in general, [8](#) identifies these main challenges:

- Communicating with people in an understandable way
- Exactness of the given information may vary
- Position information is obtained by different sensors type

The new technologies enables a new class of location-aware information systems, that link people-to-people-to-geographical-places (P3 systems). This field emerges as a result of that people in residential areas and urban work environments often have limited interaction with their geographic neighbors, and they are in regular contact with other geographically dispersed individuals. The urbanization has resulted in a shift from people-to-people-in-geographical-places towards people-to-people irrespective of local geography.

---

Location Awareness in UbiCollab
The system building efforts in location-based community information systems have lacked a firm foundation - there is little empirical knowledge of user requirements, a general conceptual framework does not exist, and the efficacy of systems design has not been empirically established[15]. We therefore use the P3-Systems framework to gain knowledge of the requirements of such systems. It is useful for us to classify our contribution within this framework, to gain more in depth knowledge about similar systems, and challenges identified within theses.

The P3-System framework
The framework organizes the design space of location-aware community systems into a 2x2 matrix of different types of system techniques. This is shown in Table 3.1.

<table>
<thead>
<tr>
<th>Design techniques</th>
<th>Synchronous communication or location awareness</th>
<th>Asynchronous communication or location awareness</th>
</tr>
</thead>
<tbody>
<tr>
<td>People-centered</td>
<td>Absolute user location</td>
<td>Uses remote awareness of current user locations</td>
</tr>
<tr>
<td></td>
<td>Uses real-time user collocation</td>
<td>Uses collocation history to exchange social information</td>
</tr>
<tr>
<td></td>
<td>Collocation/proximity</td>
<td></td>
</tr>
<tr>
<td>Place-centered</td>
<td>Use of physical places</td>
<td>Uses online representation of user’s current use of physical spaces</td>
</tr>
<tr>
<td></td>
<td>Matching virtual places</td>
<td>Uses synchronous online interaction spaces related to physical location</td>
</tr>
<tr>
<td></td>
<td>Uses asynchronous online places interactions related to physical location</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1: P3 system framework techniques summary[13].

The rows in Table 3.1 are divided into two main types:
- **People-centered** techniques use location information to support interpersonal awareness, enable communication, and identify previously unknown affinities between users.
- **Place-centered** techniques link virtual spaces to physical locations, using social information to aid place-based navigation and decision making.

People-centered techniques are the focus of our research. People-centered P3-Systems use community member’s location to improve contextualization and coordination of interactions, and enable identification of previously unidentified affinities between users[14]. The framework further subcategorizes people-centered techniques into those that represent absolute user location and those that operate in terms of user proximity or collocation. For UbiCollab we will use absolute user location in order to determine the correct Collaboration Space.

The columns in Table 3.1 are divided into two traditional types:
- **Synchronous** communication or location awareness
• **Asynchronous** communication or location awareness

Since the location of a user is constantly changing, we will design a synchronous system, using real time location information.

### 3.2 Location Technologies

In this section we will give a brief introduction to some available location technologies. These can all be used to retrieve the position of users. As mentioned in Section 2.3 this is not the focus of our task. However, in order to design and implement location-aware systems, it is useful to get an insight to the different technologies available. In the autumn project[26] we also looked into different location technologies, and for the fulfillment of this report we will repeat these in the following paragraphs.

**RFID - Radio Frequency Identification**

There are two different kinds of RFID technologies[18]:

**Passive:** Passive RFID tags operate without a battery. They reflect the RF signal transmitted to them from a reader and add information by modulating the reflected signal. Passive tags are much lighter and less expensive than active tags, offering a virtually unlimited operational lifetime. However, their read ranges are very limited.

**Active:** Active tags contain both a radio transceiver and a button cell battery to power the transceiver. Since there is an onboard radio on the tag, active tags have more range than passive tags. However these tags are larger than the passive RFID tags.

**WLAN - Wireless LAN**

The WLAN are the short range based wireless communication system. The transmitters are able to communicate with the radio object via receiver and recognize them via their network ID. These signals can be converted into distances. The transmitter can use various signal features at physical level to estimate its distance from the receiver[24].

**GSM**

GSM positioning services can be used for positioning mobile phones. This service is provided by most telephone companies. Each mobile phone with coverage is in a given cell with a unique id. The size of a cell vary from 100 m to 35 km [4]. This means that this service could be used in UbiCollab, but it is unlikely to provide the needed accuracy all the time.
GPS

If the user has a GPS receiver then this technology can be used. The GPS receiver uses radio signals from three GPS satellites [17] in order to calculate the current position of the user. This type of position gives an inaccuracy of 22 meters [2]. However the problem is that the GPS does not work for indoor positioning. Ideally we could use a combination of GPS for outdoor positioning, and for instance use of RFID for indoor positioning.

3.3 Location Models

When implementing a location aware platform for ubiquitous computing, the system must have a model of the different locations. There are many ways of storing and representing such data. One could for instance use a flat or hierarchical structure. The location data could be private, or shared. In the former one each user has their own model of the locations they are using. The latter one has just one shared model, which every user utilize. In this section we will present different models related to our work. It is said that the success of every system depends for the most part on good system design [8]. Therefore it is important to take into consideration different models, and see how they fit the system to be implemented. For instance the position in coordinates is not suitable for human users in interpreting the location of each other, but might be an adequate solution for distance calculations done by an algorithm running on a computer.

The challenges in section 3.1 are important to be aware of when the decision about proper models is made. It is quite common to distinguish between two different location models; Physical location and geographical location. Physical location is related to global geographic coordinates, where you get the position as a pair of latitude and longitude. The geographical location is used to deal with natural geographic objects, like countries, cities and buildings. These can be organized in a hierarchical structure. Following is a description of different location models within these categories, and models that capture the properties of both these models.

3.3.1 Physical location models

The physical model uses coordinates in order to define a location. Any object has a given location in the form of longitude, latitude and possibly altitude [8]. The main problem with this model is that coordinates gives little meanings to the users of such system. The location at a users office (x,y,z) gives an accurate position, but give little meaning to the users of the system. The exact location make it rather easy to calculate if a user is within an area and measuring how far he or she is from a given place. There are two different ways of describing a coordinate system that can be used. According to [7] one often distinguishes between local and global coordinate systems. A global reference system, like the one used in GPS, can be used anywhere on the planet. The local coordinate system approach is bound to a specific location, where the system is installed and configured. An example on a system that uses local coordinates is the Active Badge system [28]. In Table 3.2 pros and cons for this model are listed.

Espen Larsen Segelvik and Morten Larsen Segelvik
3. Related Research

### 3.3.2 Geographical location models

The geographical location model does not use coordinates as the physical location model\[23\]. It uses natural geographic objects in order to define a location\[8\]. The locations can be defined as a flat or hierarchical structure. The hierarchical structure requires more administration in order to organize the locations, but provides more information. For instance my office is at Telenor, Telenor is in Trondheim and Trondheim is in Norway. In this situation one could define that some applications gets access down to room level accuracy and some gets access to city level accuracy. This in turn provides the application more flexibility with regards to what information is needed. In Table 3.3 pros and cons for this model are listed.

<table>
<thead>
<tr>
<th>Pro</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Gives meaning to the users.</td>
<td>• There are no coordinates stored in this model, thus calculating distances and checking if users are within a boundary of a location can not be automated.</td>
</tr>
<tr>
<td>• Provides possibilities to store the locations in a hierarchy.</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.3: Pros and cons of geographical location models

### 3.3.3 Hybrid location models

In this model, the geographical model is extended with information from the physical model\[7\]. Adding geometric information to the model can be used to achieve higher accuracy and precision for all kinds of distance-related queries. Secondly, arbitrary geometric figures can be used, for instance, to define ranges for nearest neighbor queries. They distinguish between two types of hybrid location models. The first approach, which they call the subspace approach, stores geometric information for every modeled location. The second approach only stores geometric information for some locations, leading to partial subspaces. The benefit of the last models becomes clear when we consider a range query with a geometrically defined range. Users within a building may only know a symbolic position like room 2.1 in building...
B. Through the known geometric model, this can be translated to provide distance calculations. In Table 3.4 pros and cons for this model are listed.

<table>
<thead>
<tr>
<th>Pro</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>• The coordinates can be used to calculate distances.</td>
<td>• High maintenance.</td>
</tr>
<tr>
<td>• The logical name gives meanings to users.</td>
<td></td>
</tr>
<tr>
<td>• Provides possibility to store locations in a hierarchical structure.</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.4: Pros and cons of hybrid location models

3.3.4 Location Data Storage

The location model in a system needs to be stored. There are several different ways of storing location data. In [6] they focus on three different strategies for how to store, distribute and access location data. We will now give a brief overview of different strategies for managing the storage of location models.

1. **Infrastructure-based**: In this approach the data is stored in a globally accessible repository, allowing distributed applications to interact with their environment according to the model and their location. Within this approach the different nodes in the system must obtain their position (with GPS or other means) in order to query for different information in the model.

2. **Ad hoc**: This model relies on locally distributed data. To utilize this model, the different nodes needs to access location models based on their location, in order to obtain relevant location data.

3. **Isolated**: In this approach each node has an internal local model. Inconsistencies do not occur, since it is only one copy of the model.

3.4 Place Lab

In this section we will look into a system called Place Lab. Place Lab was built using open source and other redistributable components. The location system is based on that users define places, which can be utilized in location systems. It is useful to look into close related systems like Place Lab, to get an overview of design choices made there, and see how this can be utilized in our work.

Place Lab is developed by Intel, and the following is presented on their webpage[1]:

"Place Lab is software providing low-cost, easy-to-use device positioning for location-enhanced computing applications. Place Lab tries to provide positioning which works worldwide, both indoors and out (unlike GPS which only works well outside). Place Lab clients can determine
their location privately without constant interaction with a central service (unlike badge tracking or mobile phone location services where the service owns your location information).

The Place Lab approach is to allow devices like notebooks, PDAs and cell phones to locate themselves by listening for radio beacons such as 802.11 access points, GSM cell phone towers, and fixed Bluetooth devices that already exist in large numbers around us in the environment. These beacons all have unique or semi-unique IDs, for example, a MAC address. Clients compute their own location by hearing one or more IDs, looking up the associated beacons’ positions in a locally cached map, and estimating their own position referenced to the beacons’ positions[1].”

Architecture

The Place Lab architecture consists of three elements: Radio beacons, databases that hold information about beacons’ location, and the Place Lab clients that use this data to estimate their current location[16].

Place Lab works by listening for the transmissions of wireless networking sources such as wireless networks, bluetooth devices and GMS cell towers. All of these sources have a unique ID, which simplifies the task of calculating the client position. Clients only need to interact with beacons in order to retrieve their ID.

In order to get information from a beacon, Place Lab is dependent of knowing the location of a beacon. This information is stored in beacon databases. The data stored in these databases could be gathered from different sources such as institutions with a large amount of wireless network beacons.

The Place Lab clients use live radio observations and cached beacon locations to form an estimate of their location. The client is divided into three logical pieces:

1. **Spotters** are the eyes and ears of the client and are responsible for observing phenomenon in the physical world. A laptop client typical use an 802.11 spotter and share discovered beacon ID’s to other system components.

2. **The mapper** is responsible for providing the location of known beacons. The data provided by the mapper always contain the longitude and latitude of the beacon. The data can also contain other useful data such as the altitude and the age of the data.

3. **The tracker** uses spotter observations and associated mapper data to produce estimates of the users’s position. The tracker can use data from the mapper and spotter directly to give an estimate of the location, but can also use more extensive methods in order to give more accurate information.

3.5 Summary

Throughout this chapter we have presented research related to our work. Starting in Section 3.1 with looking at literature identifying challenges and requirements for location-aware systems. These requirements and challenges will be considered in the design. We have also looked into different location models and identified strength and weaknesses with these. It is very little information about strategies of managing the different models, but we have looked into different strategies for
storing location models. Finally we presented Place Lab, a location-aware system which shows how location can be managed with the help of user's maintaining the database. We will utilize the knowledge obtained during this chapter to supplement the following chapters.
Chapter 4

Platform Requirements

*Things should be made as simple as possible, but not any simpler*

-Albert Einstein
We will in this chapter deduce platform specific requirements for the Collaboration Space Manager and the Collaboration Instance Manager. In Chapter 2 we introduced a scenario which illustrated the usage of an application, UbiBuddy. We will start with analyzing the scenario in order to elaborate the required functionality UbiBuddy need to provide. Then we deduce high level platform requirements based on the analysis of UbiBuddy. This is followed by use cases for the Collaboration Space Manager and the Collaboration Instance Manager. Finally, both functional and non-functional requirements for the Collaboration Space Manager and the Collaboration Instance Manager are deduced from the high level requirements and the use cases.

4.1 Analysis of UbiBuddy

The scenario described in Chapter 2.4 gives us requirements regarding the functionality provided to the user by UbiBuddy. The application is built on top of the UbiCollab platform, which affects the functionality required from the platform. We will in this section analyze the different actions given in the scenario and translate them to scenario actions on the platform level. This is done in order to determine what functionality applications require from the platform. This is shown in Table 4.1.

<table>
<thead>
<tr>
<th>Scenario action</th>
<th>Actions at the platform level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action 1</td>
<td>The user is a registered in UbiCollab and has registered his tablet PC as a UbiNode.</td>
</tr>
<tr>
<td>Action 2</td>
<td>There are 2 Collaboration Instances the user is member of and one Collaboration Space defined. The user can browse existing Collaboration Instances and Collaboration Spaces.</td>
</tr>
<tr>
<td>Action 3</td>
<td>The user creates a new Collaboration Instance and adds users.</td>
</tr>
<tr>
<td>Action 4</td>
<td>The user can browse existing Collaboration Instances, and associated information, based on different criteria.</td>
</tr>
<tr>
<td>Action 5</td>
<td>Rules about awareness are associated to specific Collaboration Instances and are triggered by user location, within defined Collaboration Spaces.</td>
</tr>
<tr>
<td>Action 6</td>
<td>The user creates a new Collaboration Space. He or she can find, install, and share services.</td>
</tr>
<tr>
<td>Action 7</td>
<td>The user can add services to Collaboration Space and share the service to a Collaboration Instance.</td>
</tr>
<tr>
<td>Action 8</td>
<td>Members of a Collaboration Instance can see the Collaboration Space which users reside in and the services available.</td>
</tr>
<tr>
<td>Action 9</td>
<td>The user utilize services made available (shared) by other members of the Collaboration Instance.</td>
</tr>
<tr>
<td>Action 10</td>
<td>Available services to the members of the Collaboration Instance are changed based on users change of location.</td>
</tr>
<tr>
<td>Action 11</td>
<td>Services accessible to members of a Collaboration Instance are changed based on the presence of users in defined Collaboration Space to reflect the way they have configured their spaces.</td>
</tr>
<tr>
<td>Action 12</td>
<td>The platform support easy switching of UbiNode.</td>
</tr>
</tbody>
</table>

Table 4.1: Scenario actions at the platform level
4.2 Platform requirements

In Section 2.4.2 we identified some high level needs that the platform must support. We will in this section use these combined with the analysis of UbiBuddy in the previous section to create platform specific requirements. This is shown in Table 4.2.

<table>
<thead>
<tr>
<th>High Level Need</th>
<th>High Level Platform Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management of communities</td>
<td>• User management of Collaboration Instances</td>
</tr>
<tr>
<td></td>
<td>• Creation of Collaboration Instances</td>
</tr>
<tr>
<td></td>
<td>• Modification of Collaboration Instances</td>
</tr>
<tr>
<td></td>
<td>• Deletion of Collaboration Instances</td>
</tr>
<tr>
<td>Management of locations used for collaboration</td>
<td>• User management of Collaboration Spaces</td>
</tr>
<tr>
<td></td>
<td>• Creation of Collaboration Spaces</td>
</tr>
<tr>
<td></td>
<td>• Modification of Collaboration Spaces</td>
</tr>
<tr>
<td></td>
<td>• Deletion of Collaboration Spaces</td>
</tr>
<tr>
<td>Provide location as a resource to other components in UbiCollab</td>
<td>• Provide other components a rich interface to access location information</td>
</tr>
<tr>
<td>UbiCollab should adapt to locations</td>
<td>• Provide the correct services based on the users location</td>
</tr>
<tr>
<td>Users can share information to communities</td>
<td>• Share data in Collaboration Instances</td>
</tr>
</tbody>
</table>

Table 4.2: High Level Needs And High Level Requirements

4.3 High level use cases

In this section we will describe use cases for the different components within our task description. The components running on the platform will be realized as Managers. When we talk about Collaboration Space Manager or Collaboration Instance Manager, this refers to the service oriented components running on UbiCollab. These managers have a fixed API, which can be altered with a set of tools utilizing the interface provided by the components.
4.3.1 Collaboration Space

In this section we will create high level use cases which will help us to deduce platform specific requirements, both functional and non functional. Figure 4.1 shows a high level use case of the functionality provided by the Collaboration Space Manager. More detailed use cases will follow in this section.

![Use Case: Collaboration Space](image)

**Figure 4.1: Use Case: Collaboration Space**

**Creation of Collaboration Space**

The use case in Table 4.3 shows the creation of a Collaboration Space. The user creates a Collaboration Space by connecting to the Collaboration Space Manager through a tool. When the Collaboration Space is created the components using Collaboration Space adapts to the change.

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Creation of Collaboration Space</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Level:</strong></td>
<td>Platform</td>
</tr>
<tr>
<td><strong>Primary Actor:</strong></td>
<td>User</td>
</tr>
</tbody>
</table>

Continued on next page
4.3. High level use cases

Table 4.3 – continued from previous page

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Creation of Collaboration Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stakeholders:</td>
<td>User</td>
</tr>
<tr>
<td>Preconditions:</td>
<td>The user wants to create a new Collaboration Space</td>
</tr>
<tr>
<td>Postconditions:</td>
<td>The Collaboration Space is created</td>
</tr>
</tbody>
</table>
| Basic flow:   | 1. The user starts a tool which access the Collaboration Space Manager  
|               | 2. The user creates a new Collaboration Space at his current location  
|               | (a) The Collaboration Space is added to the list of Collaboration Spaces of that user  
|               | (b) The components using the Collaboration Space Manager adapts to the new space |
| Frequency of Occurrence: | Whenever a user wants to create a new Collaboration Space |

Table 4.3: Creation of Collaboration Space

Deletion of Collaboration Space

In Table 4.4 the deletion of a Collaboration Space is done. This is much of the same process as creating a Collaboration Space. The user connects to the Collaboration Space Manager through a tool, and the unwanted Collaboration Space is deleted.

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Deletion of Collaboration Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level:</td>
<td>Platform</td>
</tr>
<tr>
<td>Primary Actor:</td>
<td>User</td>
</tr>
<tr>
<td>Stakeholders:</td>
<td>User</td>
</tr>
<tr>
<td>Preconditions:</td>
<td>The user wants to delete a Collaboration Space</td>
</tr>
<tr>
<td>Postconditions:</td>
<td>The Collaboration Space is deleted</td>
</tr>
</tbody>
</table>
| Basic flow:   | 1. The user starts a tool to access the Collaboration Space Manager  
|               | 2. The user removes the unwanted Collaboration Space  
|               | (a) The Collaboration Space is removed from the list of the user’s Collaboration Spaces  
|               | (b) The components using the Collaboration Space Manager adapts to the change |
| Frequency of Occurrence: | Whenever a user wants to delete a Collaboration Space |

Table 4.4: Deletion of Collaboration Space

Modification of Collaboration Space

Modification of a Collaboration Space is done when some properties of the Collaboration Space change. This could be properties such as name or coordinates. This

Espen Larsen Segelvik and Morten Larsen Segelvik
is shown in Table 4.5.

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Modification of Collaboration Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level:</td>
<td>Platform</td>
</tr>
<tr>
<td>Primary Actor:</td>
<td>User</td>
</tr>
<tr>
<td>Stakeholders:</td>
<td>User</td>
</tr>
<tr>
<td>Preconditions:</td>
<td>The user wants to modify a Collaboration Space</td>
</tr>
<tr>
<td>Postconditions:</td>
<td>The Collaboration Space is modified</td>
</tr>
</tbody>
</table>
| Basic flow:   | 1. The user starts a tool to access the Collaboration Space Manager  
                 2. The user selects the Collaboration Space that needs modification  
                 3. The user modifies the appropriate data.  
                     (a) The list of Collaboration Spaces of the user is updated  
                     (b) The components using the Collaboration Space Manager adapts to the change |
| Frequency of Occurrence: | Whenever a user wants to modify a Collaboration Space |

Table 4.5: Modification of Collaboration Space

Get current Collaboration Space

In Table 4.6 the retrieval of the current Collaboration Space is shown. Different tools developed for UbiCollab can request the current Collaboration Space from the Collaboration Space Manager. Upon request from the tool the current Collaboration Space is returned.

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Get current Collaboration Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level:</td>
<td>Platform</td>
</tr>
<tr>
<td>Primary Actor:</td>
<td>UbiCollab tool</td>
</tr>
<tr>
<td>Stakeholders:</td>
<td>User</td>
</tr>
<tr>
<td>Preconditions:</td>
<td>A tool needs the current Collaboration Space of a user</td>
</tr>
<tr>
<td>Postconditions:</td>
<td>The current Collaboration Space is returned to the tool</td>
</tr>
</tbody>
</table>
| Basic flow:   | 1. The user starts a tool which needs the current Collaboration Spaces  
                 2. The tool needing the Collaboration Space request the Collaboration Space Manager for the current Collaboration Space of the user  
                 3. The current Collaboration Space is returned to the tool |
| Frequency of Occurrence: | Whenever a tool needs the current Collaboration Space of a user |

Table 4.6: Get current Collaboration Space

Location Awareness in UbiCollab
4.3. High level use cases

Automatic update of current Collaboration Space

When a user wants to automatically change the current Collaboration Space, the Collaboration Space Manager receives the coordinates of the user from a tool. Based on the coordinates, the user’s current Collaboration Space is updated. This is shown in Table 4.7.

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Automatic update of current Collaboration Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level:</td>
<td>Platform</td>
</tr>
<tr>
<td>Primary Actor:</td>
<td>User</td>
</tr>
<tr>
<td>Stakeholders:</td>
<td>User</td>
</tr>
<tr>
<td>Preconditions:</td>
<td>The user wants to automatically update his or her Collaboration Space based on his or her current location</td>
</tr>
<tr>
<td>Postconditions:</td>
<td>The current Collaboration Space is automatically updated</td>
</tr>
</tbody>
</table>
| Basic flow: | 1. The Collaboration Space Manager receives the coordinates of the user from a tool  
2. The Collaboration Space Manager determines if the user is within a Collaboration Space  
   (a) The current Collaboration Space is updated in the Collaboration Space Manager  
   (b) The components using the Collaboration Space Manager adapts to the change |
| Frequency of Occurrence: | Whenever the user wants to use automatic update of the current Collaboration Space |

Table 4.7: Automatic update of current Collaboration Space

Manual update of current Collaboration Space

The UbiCollab platform should provide the possibility to manually change the current Collaboration Space. If the user does not have a positioning device available, the user should have the possibility to set the current Collaboration Space manually. This makes the platform independent of services providing the position of users. The steps in manually changing the current Collaboration Space is shown in Table 4.8.

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Manual update of current Collaboration Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level:</td>
<td>Platform</td>
</tr>
<tr>
<td>Primary Actor:</td>
<td>User</td>
</tr>
<tr>
<td>Stakeholders:</td>
<td>User</td>
</tr>
<tr>
<td>Preconditions:</td>
<td>The user wants to manually update his or her Collaboration Space</td>
</tr>
<tr>
<td>Postconditions:</td>
<td>The current Collaboration Space is manually updated</td>
</tr>
</tbody>
</table>

Continued on next page
38

4. Platform Requirements

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Manual update of current Collaboration Space</th>
</tr>
</thead>
</table>
| Basic flow:   | 1. The user starts a tool to access the Collaboration Space Manager  
               2. The user selects the current Collaboration Space  
                  (a) The current Collaboration Space is updated in the Collaboration Space Manager  
                  (b) The components using the Collaboration Space Manager adapts to the change |
| Frequency of Occurrence: | Whenever the user wants to manually change the Collaboration Space |

Table 4.8: Manual update of current Collaboration Space

4.3.2 Collaboration Instance

In this section we will illustrate the usage of the Collaboration Instance Manager. It is important to keep in mind that this is a basic elaboration due to the scope of this thesis. The management of Collaboration Instances is needed in order to fully emphasize the endless possibilities of location awareness in UbiCollab. Figure 4.2 illustrates the basic functionality needed for the Collaboration Instance Manager.

Creation of Collaboration Instance

The use case in Table 4.9 shows the creation of a Collaboration Instance. The user creates a Collaboration Instance by connecting to the Collaboration Instance Manager through a tool. When the Collaboration Instance is created the components using Collaboration Instances is updated.

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Creation of Collaboration Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level:</td>
<td>Platform</td>
</tr>
<tr>
<td>Primary Actor:</td>
<td>User</td>
</tr>
<tr>
<td>Stakeholders:</td>
<td>User</td>
</tr>
<tr>
<td>Preconditions:</td>
<td>The user wants to create a Collaboration Instance</td>
</tr>
<tr>
<td>Postconditions:</td>
<td>The Collaboration Instance is created</td>
</tr>
</tbody>
</table>
| Basic flow:   | 1. The user starts a tool to access the Collaboration Instance Manager  
                2. The user creates a new Collaboration Instance  
                   (a) The Collaboration Instance is added to the list of Collaboration Instances of that user  
                   (b) The components using the Collaboration Instance Manager adapts to the new instance |
| Frequency of Occurrence: | Whenever a user wants to create a Collaboration Instance |

Table 4.9: Creation of Collaboration Instance

Location Awareness in UbiCollab
4.3. High level use cases

Deletion of Collaboration Instance

In Table 4.10 the deletion of a Collaboration Instance is done. This is much of the same process as creating a Collaboration Instance. The user connects to the Collaboration Instance Manager through a tool, and the unwanted Collaboration Instance is deleted.

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Deletion of Collaboration Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level:</td>
<td>Platform</td>
</tr>
<tr>
<td>Primary Actor:</td>
<td>User</td>
</tr>
<tr>
<td>Stakeholders:</td>
<td>User</td>
</tr>
<tr>
<td>Preconditions:</td>
<td>The user wants to delete a Collaboration Instance</td>
</tr>
<tr>
<td>Postconditions:</td>
<td>The Collaboration Instance is deleted</td>
</tr>
</tbody>
</table>

Continued on next page.
Table 4.10 – continued from previous page

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Deletion of Collaboration Instance</th>
</tr>
</thead>
</table>
| Basic flow:   | 1. The user starts a tool to access the Collaboration Instance Manager  
               2. The user removes the unwanted Collaboration Instance  
               (a) The Collaboration Instance is removed from the list of the user’s Collaboration Instances  
               (b) The components using the Collaboration Instance Manager adapts to the change |
| Frequency of Occurrence: | Whenever a user wants to delete a Collaboration Instance |

Table 4.10: Deletion of Collaboration Instance

Association of users to Collaboration Instances

In Table 4.11 the association of users to a Collaboration Instances is shown. This involves adding and removing users from a Collaboration Instance. A user with owner permissions connects to the Collaboration Instance Manager through a tool, and can add or remove users. Users can as they may remove themselves from any Collaboration Instance that they are members of. The use case shows a user with owner permission managing the user association to a Collaboration Instance.

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Association of users to Collaboration Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level:</td>
<td>Platform</td>
</tr>
<tr>
<td>Primary Actor:</td>
<td>Collaboration Instance Owner</td>
</tr>
<tr>
<td>Stakeholders:</td>
<td>User</td>
</tr>
<tr>
<td>Preconditions:</td>
<td>An owner wants to add/remove a user from a Collaboration Instance</td>
</tr>
<tr>
<td>Postconditions:</td>
<td>The user is added/removed</td>
</tr>
</tbody>
</table>
| Basic flow:   | 1. A user with owner permission starts a tool to access the Collaboration Instance Manager  
               2. The user adds/removes a user  
               (a) The user is added/removed from the Collaboration Instance user list  
               (b) The components using the Collaboration Instance Manager adapts to the change |
| Frequency of Occurrence: | Whenever a user is added/removed |

Table 4.11: Association of users to Collaboration Instances

Modification of Collaboration Instance

Modification of Collaboration Instances is done when some properties of a Collaboration Instance change. This could be properties such as name and other properties related to the Collaboration Instance. This is shown in Table 4.12

Location Awareness in UbiCollab
4.3. High level use cases

### Modification of Collaboration Instance

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Modification of Collaboration Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level:</td>
<td>Platform</td>
</tr>
<tr>
<td>Primary Actor:</td>
<td>User</td>
</tr>
<tr>
<td>Stakeholders:</td>
<td>User</td>
</tr>
<tr>
<td>Preconditions:</td>
<td>The user wants to modify a Collaboration Instance</td>
</tr>
<tr>
<td>Postconditions:</td>
<td>The Collaboration Instance is modified</td>
</tr>
</tbody>
</table>
| Basic flow:   | 1. The user starts a tool to access the Collaboration Instance Manager  
                2. The user selects the Collaboration Instance that needs a modification  
                3. The user modifies the appropriate data  
                    (a) The list of Collaboration Instances of the user is updated  
                    (b) The components using the Collaboration Instance Manager adapts to the change |
| Frequency of Occurrence: | Whenever a user wants to modify a Collaboration Instance |

Table 4.12: Modification of Collaboration Instance

### Sharing of data in a Collaboration Instance

The sharing of data is shown in Table 4.13. This is done through a tool accessing the Collaboration Instance Manager. When data is shared in a Collaboration Instance, it becomes available for all members of that Collaboration Instance. This can be various information, such as services used for collaboration, or information about availability of users, and the current Collaboration Space of users.

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Sharing of data in a Collaboration Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level:</td>
<td>Platform</td>
</tr>
<tr>
<td>Primary Actor:</td>
<td>User</td>
</tr>
<tr>
<td>Stakeholders:</td>
<td>User</td>
</tr>
<tr>
<td>Preconditions:</td>
<td>The user wants to share data in a Collaboration Instance, thus making it available for the other users in the Collaboration Instance</td>
</tr>
<tr>
<td>Postconditions:</td>
<td>Data is shared in Collaboration Instance, and made available for other members</td>
</tr>
</tbody>
</table>
| Basic flow:   | 1. The user starts a tool to access the Collaboration Instance Manager  
                2. The user selects the data he or she wants to share  
                3. The data is shared in the Collaboration Instance Manager via the API provided by the Collaboration Instance Manager  
                    (a) The new data is made available to all users in the Collaboration Instance |
| Frequency of Occurrence: | Whenever a user wants to share data |

Table 4.13: Sharing of data in a Collaboration Instance

Espen Larsen Segelvik and Morten Larsen Segelvik
Get Collaboration Instance Data

Users in a Collaboration Instance should have the possibility to access information other users has shared in the Collaboration Instance. This is done with an application utilizing the API provided by the Collaboration Instance Manager. The steps of this process is shown in Table 4.14.

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Get Collaboration Instance data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level:</td>
<td>Platform</td>
</tr>
<tr>
<td>Primary Actor:</td>
<td>User</td>
</tr>
<tr>
<td>Stakeholders:</td>
<td>User</td>
</tr>
<tr>
<td>Preconditions:</td>
<td>The user wants to view data in a Collaboration Instance shared by other users</td>
</tr>
<tr>
<td>Postconditions:</td>
<td>The shared data in a Collaboration Instance is shown</td>
</tr>
</tbody>
</table>
| Basic flow:                   | 1. The user starts a an application which shows the Collaboration Instances  
                                     2. The user selects the Collaboration Instance that he or she wants data from  
                                     3. The data shared in the Collaboration Instance is shown to the user |
| Frequency of Occurrence:      | Whenever a user wants to view data shared in a Collaboration Instance                            |

Table 4.14: Get Collaboration Instance data

### 4.4 Platform specific requirements

In the previous section we identified some high-level use cases and in Section 2.4.2 we identified requirements the UbiCollab platform should fulfill. From this we will deduce the platform specific requirements, both functional and non functional.

#### 4.4.1 Functional requirements

Functional requirements describe what functionality the platform should provide regarding the management of Collaboration Spaces and Collaboration Instances. From Section 4.2 and Section 2.4.2 we have deduced the requirements listed in Table 4.15.

<table>
<thead>
<tr>
<th>Id</th>
<th>Priority</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS-FREQ1</td>
<td>High</td>
<td>Create Collaboration Space</td>
</tr>
<tr>
<td>CS-FREQ2</td>
<td>High</td>
<td>Delete Collaboration Space</td>
</tr>
<tr>
<td>CS-FREQ3</td>
<td>High</td>
<td>Modify Collaboration Space</td>
</tr>
<tr>
<td>CS-FREQ4</td>
<td>High</td>
<td>Connect Services to Collaboration Space</td>
</tr>
<tr>
<td>CS-FREQ5</td>
<td>High</td>
<td>Provide an interface in order to retrieve Collaboration Space information</td>
</tr>
</tbody>
</table>

Continued on next page
4.4. Platform specific requirements

Table 4.15 – continued from previous page

<table>
<thead>
<tr>
<th>Id</th>
<th>Priority</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS-FREQ6</td>
<td>High</td>
<td>Provide functionality in order to calculate the current Collaboration Space of a user</td>
</tr>
<tr>
<td>CS-FREQ7</td>
<td>High</td>
<td>Automatic update of Collaboration Space based on location</td>
</tr>
<tr>
<td>CI-FREQ1</td>
<td>High</td>
<td>Add Collaboration Instance</td>
</tr>
<tr>
<td>CI-FREQ2</td>
<td>High</td>
<td>Delete Collaboration Instance</td>
</tr>
<tr>
<td>CI-FREQ3</td>
<td>High</td>
<td>Modify Collaboration Instance</td>
</tr>
<tr>
<td>CI-FREQ4</td>
<td>High</td>
<td>Associate users to a Collaboration Instance</td>
</tr>
<tr>
<td>CI-FREQ5</td>
<td>High</td>
<td>Share data in Collaboration Instance</td>
</tr>
</tbody>
</table>

Table 4.15: Functional requirements

**CS-FREQ1**

The creation of Collaboration Spaces is important in order to be able to create new virtual representations of physical locations. Each user should be able to create their own Collaboration Spaces which corresponds with their personal perception of that place. Thus this requirement has priority high.

**CS-FREQ2**

If a Collaboration Space is no longer needed, the users should have the opportunity to delete the Collaboration Space. This is important in order to only have the Collaboration Spaces actually needed for the user available, therefore this requirement has priority high.

**CS-FREQ3**

Collaboration Space changes, for instance if a user gets a new office. The user should have the opportunity to modify his Collaboration Space in order to compensate for the physical changes. However there are not only physical changes that require modifications. The user might want to change the name of the Collaboration Space, and therefore needs to modify the Collaboration Space. This is important in order to keep the Collaboration Spaces up to date, therefore this requirement has priority high.

**CS-FREQ4**

Collaboration can take place anywhere, for instance in your home, at you office etc. At these places you often have artifacts that can be used in collaboration, like displays, loudspeakers etc. If we allow these services to be connected to Collaboration Spaces, the platform can easily adapt to change of location. This is important since there is no use to have active services for instance at home, when the user is at another place. Thus this requirement has priority high.
CS-FREQ5

The UbiCollab platform should be location-aware. Therefore different components should be able to retrieve location information, in order to adapt to changes in location. Therefore it is important to have a fixed API for informing other components of the platform of the current Collaboration Space. Thus this requirement has priority high.

CS-FREQ6

Based on coordinates, the Collaboration Space Manager should be able to calculate the current Collaboration Space of a user. This minimizes the user interaction needed when changing location. Thus this has priority high.

CS-FREQ7

The UbiCollab platform should be able to support automatic update of current Collaboration Space based on coordinates. The platform should also support the use of different position technologies in order to calculate the current Collaboration Space. It is important that this is implemented in a flexible way, in order to support new and emerging position technologies. Thus this requirement has priority high.

CI-FREQ1

The Collaboration Instance Manager should have an easy way of creating a new Collaboration Instance. If this becomes cumbersome for the user, the risk is that the different users will utilize other means of collaboration instead of utilizing UbiCollab. Therefore this requirement has high priority.

CI-FREQ2

During everyday life users communities evolve. It is important to have the possibility to remove a community when a collaboration period is over. Therefore it is important to provide an interface for deleting Collaboration Instances.

CI-FREQ3

Collaboration Instances are a representation of communities. Communities evolve over time, and this should be accounted for in the Collaboration Instance Manager. The Collaboration Instance Manager should have a flexible way of modifying different Collaboration Instances, so that they easily keep up with the ever changing environment. Thus this has priority high.
4.4. Platform specific requirements

CI-FREQ4

The creation of a Collaboration instance itself is not enough. People forms communities, thus it should be an easy way of associate users to a Collaboration Instance. Thus this has priority high.

CI-FREQ5

When collaborating with dispersed people it is important to have a way of sharing relevant data. Data can span from pictures to services needed to perform a task. It is important to have a generic method for sharing necessary data within a community. Thus this has priority high.

4.4.2 Non-functional requirements

The non-functional requirements are requirements concerning all functionalities that will run in the periphery of the system. These functions are not visible to the user, but are still needed to perform the intended functionality of the system. The non-functional requirements are shown in Table 4.16.

<table>
<thead>
<tr>
<th>Id</th>
<th>Priority</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>N-FREQ1</td>
<td>High</td>
<td>Scalability</td>
</tr>
<tr>
<td>N-FREQ2</td>
<td>High</td>
<td>OS independency</td>
</tr>
<tr>
<td>N-FREQ3</td>
<td>High</td>
<td>End user programming</td>
</tr>
<tr>
<td>N-FREQ4</td>
<td>High</td>
<td>Extendibility</td>
</tr>
<tr>
<td>N-FREQ5</td>
<td>High</td>
<td>Service oriented architecture</td>
</tr>
</tbody>
</table>

Table 4.16: Non-functional requirements

N-FREQ1

UbiCollab will be used in a variety of communities, spanning from small groups to quite large groups. It is intended to help collaboration in a variety of ways, and therefore the number of users and communities is hard to foresee at this time. Thus it is important that the different platform components are scalable in order to perform well within a wide range of different collaborative settings. To do so the network traffic must be kept at a minimum level and the platform elements should when possible be distributed in order to avoid bottlenecks. Due to a potential large amount of users within a community, and large numbers of communities, this requirement has priority high.

N-FREQ2

Due to the distributed nature of the platform and its services, the UbiCollab platform should be OS independent. The platform services will probably be running on a wide variety of devices with different operating systems. Thus providing an...
OS independent platform is extremely important. Different devices have different operating systems, and the platform should be implemented to support as many as possible, thus it has priority high.

N-FREQ3

The end-users will not interact directly with the platform components, but the API provided by the platform will affect the applications built on top of it. This will thereby influence the end-user experience. Thus it is important to have a rich API specified, that each application can use to perform the intended task. This has priority high.

N-FREQ4

The numbers of different collaboration settings are hard to foresee at this stage of the development. It is also difficult to take into account every variety of variables needed to perform a given collaboration task. New ubiquitous technologies are emerging rapidly. Thus during the lifetime of the system new and emerging technologies most probably will appear. The UbiCollab platform should easily be extendible to support these new and emerging technologies. If the platform is hard to extend, the user will probably not bother to extend the system, which eventually will result in an outdated platform. At this early stage of the UbiCollab project it is therefore extremely important with an architecture that is easily extended, and the priority of this requirement is high.

N-FREQ5

One of the aims for UbiCollab is to develop a service oriented architecture. This has implication both on the design and implementation of the platform. In practice this means that each service provided by the platform should be designed as a standalone service. However the functionality provided should be the same as if it was implemented otherwise. Therefore the fulfillment of this requirement has priority high.
Chapter 5

Design

Design is a plan for arranging elements in such a way as best to accomplish a particular purpose

-Charles Eames
In this chapter we will design the different components of UbiCollab within our work. The main focus of our work is the Collaboration Space Manager, but we will create a simple design of the Collaboration Instance Manager. The Collaboration Instance Manager will be used in a demonstrator in Chapter 7 and will illustrate our work on the Collaboration Space Manager. We will start with describing the technical background affecting our design, followed by a description of the UbiCollab architecture. Finally, we design the Collaboration Instance Manager and the Collaboration Space Manager.

5.1 Background

Working within the UbiCollab project gives us some guidelines that we need to follow. The first and most important one is that the platform should be developed in a Service Oriented Architecture. More specific, this means that each component in the platform is developed as a service, providing an interface to the components functionality. The second guideline is that the services should be developed as OSGi bundles.

5.2 UbiCollab architecture

The UbiCollab follows the Service Oriented Architecture approach, where the entire platform is developed as independent components deployed as services. Each component of the platform covers a specific area of responsibility in UbiCollab. By mixing different components together, the platform can be tailored to a specific application. The components needed are deployed on the device UbiCollab will be running on.

Figure 5.1 shows the different components in UbiCollab. These components will operate "per user". i.e. the information is distributed among the users, and each user in UbiCollab will have their own instance of each of these components. A description of these components and our work within these are listed in Table 5.1.

<table>
<thead>
<tr>
<th>Component</th>
<th>Responsibility</th>
<th>Our work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Discovery Manager</td>
<td>Is responsible for finding Services in the user's surrounding. It gets a service query and returns a Service URI referring to a Service Advertisement. It relies on Service Discovery Plug-ins to find the URI through native discovery protocols (e.g. through RFID, Bluetooth, SLP, UDDI).</td>
<td>None, this component will be developed by another master thesis student</td>
</tr>
<tr>
<td>Service Domain Manager</td>
<td>Is responsible for installing service proxies, managing data related to service proxies, and answering queries about availability of service proxies.</td>
<td>This component will be developed by another master thesis student[1]. We will integrate our work with this component</td>
</tr>
</tbody>
</table>

Location Awareness in UbiCollab

Continued on next page
### Table 5.1: Description of the different components in UbiCollab

<table>
<thead>
<tr>
<th>Component</th>
<th>Responsibility</th>
<th>Our work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collaboration Instance Manager</td>
<td>Is responsible for managing a user’s Collaboration Instances and the data shared in each Collaboration Instance. The Collaboration Instance Manager is also responsible for enabling the sharing of Collaboration Instance object among Collaboration Instance members distributed across a network.</td>
<td>Create a simple design and implementation, to demonstrate the functionality of the UbiCollab platform</td>
</tr>
<tr>
<td>Collaboration Space Manager</td>
<td>Is responsible for creating Collaboration Spaces, managing existing Collaboration Spaces, querying Collaboration Spaces based on point coordinates and locate points in Collaboration Spaces.</td>
<td>Create a thorough design and implementation of this component.</td>
</tr>
<tr>
<td>Service Ontology Manager</td>
<td>Is responsible for maintaining ontology data, and providing search and query mechanisms for UbiCollab service ontology. Currently only in design phase.</td>
<td>None, this component will be developed by another master thesis student</td>
</tr>
<tr>
<td>Service Composer</td>
<td>Is responsible for maintaining/validating/managing service compositions, for executing service compositions, and for sharing of service compositions.</td>
<td>None, this component will be developed by another master thesis student</td>
</tr>
<tr>
<td>Identity Manager</td>
<td>Is responsible for managing (creating, defining, deleting) virtual identities for a user. Currently only in design phase.</td>
<td>None, will be designed on a later stage of this project</td>
</tr>
</tbody>
</table>

5.2.1 Tools

Tools are used to alter data in the different platform components. Both the Collaboration Instance Manager and the Collaboration Space Manager have tools in order to do this. If an application wants to change data in the respectively components, it needs to utilize the tools provided. In the following we will describe three tools needed for managing the Collaboration Space Manager, Collaboration Instance Manager and publishing of information in the Collaboration Instances.

**Collaboration Space Manager Tool**

The Collaboration Space Manager Tool provides a graphical user interface for the user. This tool enables the user to use the functionality provided by the Collaboration Space Manager. The user can create, modify and delete Collaboration Spaces with this tool.
Collaboration Instance Manager Tool

The Collaboration Instance Manager Tool provides a graphical user interface for the user. This tool enables the user to create, modify and delete Collaboration Instances. It also provides the functionality to add and remove users from a Collaboration Instance. A user can either be an owner or a regular user. Only owners of a Collaboration Instance can add and delete other users.

Collaboration Instance Updater Tool

This tool is for sharing information in a Collaboration Instance. It gathers information from other platform components and shares this. When the information is shared, this information becomes available for every participant of the Collaboration Instance.
5.3. Collaboration Instance Manager

5.2.2 Applications

UbiCollab applications provide views into Collaboration Instances, and can visualize the information. The application will therefore only communicate with the Collaboration Instance Manager at the platform level. Applications can however use different tools provided by the different components at the platform, to alter, create and remove data.

5.2.3 UbiHome

The UbiHome server provides persistent data storage to the components of the UbiCollab platform. This is also not the focus of our work, but for demonstrating purposes, we will implement a working version of this component. The main idea of the UbiHome server is to provide the Collaboration Space Manager and the Collaboration Instance Manager with persistent data storage. All the personal configuration and data of the user is stored here. This enables users to use any UbiCollab enabled device and retrieve the same configuration.

5.3 Collaboration Instance Manager

The Collaboration Instance Manager handles the management of Collaboration Instances. This is not the main focus of our work, but in order to demonstrate the concept of Collaboration Spaces, we need a basic design of the Collaboration Instance Manager. The main focus of this component is to provide the basic functionality of community management. From the platform requirements we found in Chapter 4, this component needs to provide the following functionality:

- User management: Add and remove users
- Modify Collaboration Instance: Administrative task related to different properties of a Collaboration Instance.
- Publish data: Used to share data to aid collaboration.

Granularity

One property connected to Collaboration Instances is the level of abstraction of information shared. We choose to let the users define their level of abstraction by providing a granularity level, which ranges from 0-100. If a user participate in a Collaboration Instance with granularity level 50, all information tagged with a certain granularity level between 51-100 will not be available in this Collaboration Instance. Introducing this property in a Collaboration Instance provides fine grained control over the sharing of information.

5.3.1 Sequence Charts

In this section we will describe different sequence charts related to the Collaboration Instance. These shows the interaction between different components related to...
the Collaboration Instance Manager. We base these sequence charts on platform requirements in the previous chapter.

Create Collaboration Instance

In Figure 5.2 the creation of a Collaboration Instance is shown. An application or a user starts the Collaboration Instance Tool. Then the Collaboration Instance Tool sends the new Collaboration Instance information to the Collaboration Instance Manager, which in turn updates the UbiHome server.

Delete Collaboration Instance

In Figure 5.3 the deletion of a Collaboration Instance is shown. An application or a user starts the Collaboration Instance Manager Tool. Then the tool sends the delete command to the Collaboration Instance Manager. The Collaboration Instance is then deleted from the manager, and the UbiHome server is updated.

Add User to Collaboration Instance

Figure 5.4 shows the sequence of events when a user is added to a Collaboration Instance. An owner of a Collaboration Instance starts the Collaboration Instance Manager Tool, either via an application or starting it manually. Then he or she adds the user, which is sent to the Collaboration Instance Manager. The manager is updated, and sends this information to the UbiHome server.
Delete User from Collaboration Instance

Figure 5.5 shows the deletion of a user in a Collaboration Instance. The Collaboration Instance Manager Tool is started through an application or by the user. The tool sends the information to the Collaboration Instance Manager, which removes the user from its internal model and on the UbiHome Server.

Modify Collaboration Instance

In order to modify a Collaboration Instance, users must utilize the Collaboration Instance Manager Tool. This can be started either through an application or by a user. When the user has modified the Collaboration Instance in the tool, the Collaboration Instance Manager and the UbiHome server is updated. The sequence chart for this action is shown in Figure 5.6.
Publish data in a Collaboration Instance

Users can publish data in the Collaboration Instance with the use of a publisher tool. This tool uses the API provided by the Collaboration Instance Manager, and when data is published, it is synchronized with the other users. The sequence chart for this sequence is shown in Figure 5.7.

Figure 5.7: Publish data in a Collaboration Instance Sequence

Get published Collaboration Instance data

Applications can utilize the data shared in a Collaboration Instance. To get a hand on the shared information, the application must call the getPublishedData method provided by the Collaboration Instance Manager. The sequence chart for this action is shown in Figure 5.8.

Figure 5.8: Get published Collaboration Instance data Sequence

5.3.2 API

Based on the sequence charts, we have created an API for the Collaboration Instance Manager. This is shown in Figure 5.9 and a brief description of the methods is given below.

- **login**: A method for retrieving the users Collaboration Instances from the UbiHome server.
- **getCIData**: A method for retrieving data from the Collaboration Instance Manager.
- **createCI**: A method for creating a new Collaboration Instance.
5.4. Collaboration Space Manager

- **deleteCI**: A method for deleting a Collaboration Instance. This requires that the user is a owner of the Collaboration Instance. If the user is not a owner, the user will only be removed from the Collaboration Instance.

- **modifyCI**: A method for modifying a Collaboration Instance.

- **addUser**: A method for adding a user to a Collaboration Instance.

- **deleteUser**: A method for deleting a user from a Collaboration Instance.

- **publishData**: A method for publishing various data. This can be services, current Collaboration Space or other relevant data that a user wants to share within a Collaboration Instance.

- **getPublishedData**: A method for the retrieval of shared data within a Collaboration Instance.

![Collaboration Space Manager API](image)

Figure 5.9: Collaboration Space Manager API

5.4 Collaboration Space Manager

The Collaboration Space Manager is the main focus of our work. This handles the management of spaces, which include the following:

- Creation of Collaboration Spaces
- Modification of Collaboration Spaces
- Deletion of Collaboration Spaces
- Mapping of user to Collaboration Space
- Mapping of physical location to Collaboration Space

A Collaboration Space is a virtual representation of physical locations. The Collaboration Space Manager should handle this mapping in addition to provide an API for management of these. The API should provide the ability to add, modify and delete Collaboration Spaces. If a user does not have a device to provide his or her current coordinates, the Collaboration Space Manager should provide an API to update the current Collaboration Space manually.
5.4.1 Definition of a Collaboration Space

A Collaboration Space is defined by several properties. When representing a physical location as a Collaboration Space, the physical properties must be represented. These properties are listed in Table 5.2. These locations can be represented in various ways, for instance in a flat or a hierarchical structure. This will be elaborated in the next section.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>A descriptive name of the physical location.</td>
</tr>
<tr>
<td>Shape</td>
<td>Describes the shape of the location. This could for instance be a square or a circle.</td>
</tr>
<tr>
<td>Coordinates</td>
<td>Describes where the shape is located.</td>
</tr>
<tr>
<td>Characteristics</td>
<td>Defines if the Collaboration Space is mobile or static. Mobile Collaboration spaces does not have fixed coordinates, i.e. a car is a Collaboration Space, but changes coordinates as the user drive the car.</td>
</tr>
</tbody>
</table>

Table 5.2: Description of Collaboration Space properties

5.4.2 Location Model

As elaborated in Chapter 3, there are many ways of organizing locations. The different location models have both positive and negative sides. However, the model that will be used in UbiCollab requires both interaction between humans and computers, thus we feel that the Hybrid location model is the most appropriate approach. Using this approach, locations can easily be arranged in a hierarchical structure that makes sense for the users. The drawback of using a hierarchical structure is that it requires more management when defining a Collaboration Space, thus requiring more interaction from the user. Even though this requires more interaction, we feel that this way of organizing locations suits UbiCollab better. According to [25] in real world scenarios we usually find hierarchical structures, e.g., a room is inside a building, a building is in a city, a city is in a country etc. Since we will be modeling real world locations, we feel that capturing this aspect is important. Extending the hierarchy with a shape and coordinates makes it possible to easily calculate distances, which can be used for querying location relevant information, and mapping users to Collaboration Spaces.

5.4.3 Personal vs. shared Collaboration Spaces

There are mainly two different ways of managing Collaboration Spaces. One way is to have a shared register of Collaboration Spaces, with shared names. The second alternative is that each user has personal Collaboration Spaces, defined by the users themselves.

Shared Collaboration Spaces

In the shared model, all Collaboration Spaces are shared among the users. This can either be maintained by the users, or by an administrative authority responsible for...
5.4. Collaboration Space Manager

the system. As described in Chapter 3, Place Lab is a system using this approach, where the users are responsible for maintaining the model. The users have every location in the system available. These shared locations must be given a name that everyone can use, such as "Room 207". The problem with this naming strategy is that it becomes impersonal and gives little information. If a user is at home, an appropriate name for Collaboration Space would be "My home" instead of the street address. Another problem with this model is that management of the Collaboration Spaces becomes difficult. There must be some access rules in order to add, delete and modify Collaboration Spaces in the system. The last problem we see with this model is that the amount of locations defined, can become large. If a user wants to set his or her current Collaboration Space manually, the task can be cumbersome if the users can choose between hundreds of places. Besides the negative effects of introducing this approach, there are benefits. When arriving at a new location, users can easily use the Collaboration Spaces already defined in system. Thus using this approach minimizes the effort for using location relevant information.

Personal Collaboration Spaces

In the second alternative, all the users have their own record of their Collaboration Spaces. The user has the task of defining Collaboration Spaces and the management of these. If many users use the same Collaboration Space, all the users must define the same location as a Collaboration Space. This can be solved by letting the users share common Collaboration Spaces. I.e., retrieving a copy of other Collaboration Spaces, which they can manage as they like. The main benefits of this model are that the users have full control of their Collaboration Spaces, and any modification of a Collaboration Space does not affect other users. The users only have the Collaboration Spaces he or she needs.

Discussion

The two different strategies have different strengths and weaknesses. The shared Collaboration Space model is probably the strategy that requires the smallest amount of configuration and management for the users. Even though, we feel that the flexibility that the personal Collaboration Space strategy gives is more suitable for UbiCollab. Another important aspect is that UbiCollab is a platform to aid collaboration when the users are dispersed. Even though collaboration can take place everywhere, the typical collaboration takes place in fixed places such as home and work. Letting the users define their own Collaboration Spaces makes the location model more relevant for each user. This also reduces the maintenance cost for the UbiCollab platform. Each user is responsible for keeping their location model up to date. Since we are letting the users define their Collaboration Spaces, each Collaboration Space can have different meanings for different users. Pedro’s home will be defined as "Home" by Pedro. If Lola has this place defined as a Collaboration Space, she would probably define this as "Pedro’s Home", thus aiding location awareness. Therefore we choose to let the users of UbiCollab have their own location model. When the user defines a new Collaboration Space, he or she is also responsible to arrange the space in their hierarchical structure. The users can add the new Collaboration Space as a "root space", or select a parent. In Chapter 3, we looked into how Place Lab utilizes user interaction in order to define locations. In this system each
location is stored in a database. We feel that letting the users define places related to collaboration is a useful approach. Taking into consideration that locations have different meanings for each user, a personal location model is desirable.

5.4.4 Mapping from coordinates to Collaboration Spaces

The mapping between physical locations and Collaboration Spaces are shown in Figure 5.10. It is important to mention that a physical location can have different virtual representation according to the different users and different usage. If a user has a home office, the home then has two different virtual representations (home and office).

The mapping from a physical location to a virtual representation is a tricky process. Regarding the Collaboration Spaces there are two different types of mapping. First is the mapping of a physical location to a Collaboration Space. Second is the mapping of individual users to the different Collaboration Spaces. Both these mappings require that the coordinates of a physical location is stored within the Collaboration Space Manager. These could be retrieved from a GPS device or other position techniques. A location usually consists of a square, circle or other geometric shapes. When a user moves around, the user’s coordinates must be checked to see if his or her current position is within any boundary of the geometric boundaries defined by the user. Another property of Collaboration Spaces that makes this process even trickier is that some Collaboration Spaces can be mobile. For instance, a user can define his car as a Collaboration Space. Obviously this Collaboration Space does not have fixed coordinates, and must be treated differently form static Collaboration Spaces.

Figure 5.10: Mapping between physical locations and Collaboration Spaces
5.4.5 Granularity issues

Different communities demand different information regarding the detail of Collaboration Space information. One community only needs to know that you are in Trondheim, but another requires knowing that you are in your office. To solve this issue, the users can arrange the different Collaboration Spaces in a hierarchical structure, either by creating the new Collaboration Space as a "root node" or select a parent node. This principle is shown in Figure 5.11. When defining a Collaboration Space the user must provide a granularity level. This is used when sharing a Collaboration Space with other users in a Collaboration Instance to raise the level of location awareness. The granularity level ranges from 0-100. As illustrated in Figure 5.11, the user can be in his office. If he decides to share his or her Collaboration Space in a Collaboration Instance with a granularity level 0, Trondheim will show as the current Collaboration Space. Providing a granularity level in each Collaboration Space makes it easy to configure the level of abstraction shared among other participants in a flexible way.

![Collaboration Space Hierarchy](image)

Figure 5.11: Collaboration Space Hierarchy

5.4.6 Sequence Charts

In this section we will describe different sequence charts related to the Collaboration Space Manager. These shows the interaction between different components related to the Collaboration Space Manager. We base these sequence charts on platform requirements in the previous chapter. In order to demonstrate some of the functionality, the usage of a GPS device is required. We assume that we have a Service Discovery Manager that has an API for retrieving such a device.
Add Collaboration Space

In Figure 5.12 the sequence of actions for adding a collaboration space is shown. The application or the user starts the Collaboration Space Manager Tool. The user is shown a graphical user interface, where he or she can define a new Collaboration Space. The Collaboration Space Manager receives the new information, which in turn is sent to the UbiHome server for persistent storage.

![Figure 5.12: Add Collaboration Space Sequence](image)

Delete Collaboration Space

In Figure 5.13 the deletion of a Collaboration Space is shown. The application or the user starts the Collaboration Space Manager tool. The user selects from the user interface the Collaboration Space he or she wants deleted. The Collaboration Space Manager then deletes the Collaboration Space from its internal model, and from the UbiHome server. On the next request for Collaboration Spaces this one is removed.

![Figure 5.13: Delete Collaboration Space Sequence](image)

Modify Collaboration Space

In Figure 5.14 modification of Collaboration Space is shown. The application or the user starts the Collaboration Space Manager Tool. The user then modifies a Collaboration Space, and the updated information is sent to the Collaboration...
5.4. Collaboration Space Manager

Space Manager. The internal model for the Collaboration Spaces is updated, and sent to the UbiHome server.

![Diagram showing the modify collaboration space sequence]

**Figure 5.14: Modify Collaboration Space Sequence**

**Get Current Collaboration Space**

Figure 5.15 illustrates how the API from the Collaboration Space Manager can be used to retrieve the current Collaboration Space of a user. The Collaboration Space Manager requires a granularity value. The granularity is described in Section 5.4.5. In order to retrieve data in the Collaboration Space Manager, a tool is used. This tool can either be the Collaboration Space Manager Tool described in Section 5.2.1 or another tool created for a particular purpose.

![Diagram showing the get current collaboration space sequence]

**Figure 5.15: Get Current Collaboration Space Sequence**

**Get Nearest Collaboration Space**

In Figure 5.16, the calculation of the nearest Collaboration Space is shown. This method requires latitude, longitude parameters as well as the granularity level. Latitude and longitude are typically retrieved from some kind of position technologies. As for the get current Collaboration Space sequence, this could be done with the Collaboration Space Manager Tool, or other tools created for a particular purpose.

![Diagram showing the get nearest collaboration space sequence]
Set Current Collaboration Space

Figure 5.17 illustrates how a user can utilize the tool developed for each component. The user can either start the tool manually, or through an application. In this sequence chart, the tool provided retrieves all the Collaboration Spaces for the current user. The user then manually selects the current Collaboration Space and this is set as the current one.

Automatic publishing of Collaboration Space in Collaboration Instance

Figure 5.18 shows how automatic publishing of Collaboration Spaces works. The application sends the instruction that automatic publishing is desired. The tool provided, asks the Service Discovery Manager for a GPS device. Then the tool retrieves the position from the positioning device. Then the tool asks for the nearest Collaboration Space based on the current position retrieved from the position device, and the granularity level in each Collaboration Instance. The current Collaboration Space of the user is then published in the Collaboration Instances.
5.4.7 API

Based on the sequence charts in the previous Section and the requirements in Chapter 4, we will propose an API for the Collaboration Space Manager. The API for the Collaboration Space Manager is shown in Figure 5.19, and a brief description of the methods are given below.

- **login**: A method for retrieving the personal Collaboration Space(s) of the user logging in.
- **getCSData**: A method for retrieving every space in a Collaboration Space Manager.
- **addCS**: A method for adding a new Collaboration Space. The new space is added to the Collaboration Space Manager, and the UbiHome server.
- **deleteCS**: A method for deleting a Collaboration Space. This method removes the Collaboration Space, both from the internal model in the Collaboration Space Manager and the UbiHome Server.
• **modifyCS**: A method for modifying a Collaboration Space. By updating a Collaboration Space, the internal model of the Collaboration Space Manager and the UbiHome server will be updated.

• **getCurrentCS**: A method for retrieving the current Collaboration Space of a user. This requires a granularity. If the requesting granularity is not high enough, the returned Collaboration Space is a node further up in the hierarchical structure.

• **setCurrentCS**: A method for setting the current Collaboration Space.

• **getNearestCS**: A method for retrieving the nearest Collaboration Space based on the coordinates of a user. This requires GPS formatted coordinates, and will return the closest Collaboration Space, and the distance to this one.

In Chapter 3 we identified four requirements regarding the API for the component responsible for location awareness. Our API conforms to most of these requirements. However, we have chosen not to elaborate the requirement regarding accuracy. This is not an important aspect in order to fulfill the main goal of this thesis, and taking this into account is a huge task. Our design is not committed to only one standard, and the Collaboration Space Manager is designed to use any kind of positioning technology. The Collaboration Space Manager is not dependent of any underlying technology, due to the usage of independent positioning services available in the platform.

## 5.5 Putting it all together

In this section we will look into how it all comes together. Figure 5.20 shows a sequence chart of the login procedure. Application uses a login tool, which logs in on the different platform components. Relevant for our task is the login sequence for the Collaboration Space Manager and Collaboration Instance Manager. When these get a login request, they respond with retrieving data from the UbiHome server, where persistent data storage is provided. When these components are up and running, an updater tool is started. This tool publishes the current Collaboration Space of a user in the different Collaboration Instances, based of predefined rules. The application can then start requesting data from the Collaboration Instance Manager.
Figure 5.20: Login Sequence
5. Design

Location Awareness in UbiCollab
Chapter 6

Implementation

Shoot for the moon. Even if you miss it you will land among the stars.
-Les Brown.
We have implemented the components described in the design. The main focus of our implementation has been on the Collaboration Space Manager. We have also implemented a simple version of the Collaboration Instance Manager. In order to manage these components, we have implemented one tool for each of them. In addition to these components we have implemented a version of the UbiHome server, which provides an interface for storing persistent data. This chapter starts with an overview of the implementation, which is followed by a detailed description. Finally we compare the implementation with regards to the design.

6.1 Implementation overview

We have developed several platform components which are shown in Figure 6.1. The light blue boxes are our contributions and the boxes colored gray are components developed by other Master thesis students and integrated with our work\[12\].

![Implementation overview](image)

Figure 6.1: Implementation overview

We will in the following sections describe the implementation of the different components, starting with the Collaboration Space Manager and the Collaboration Space Manager tool(csmTool), followed by the Collaboration Instance Manager and the Collaboration Instance Manager Tool(cimTool). Then the tool connecting the Collaboration Space Manager and the Collaboration Instance Manager together will be

Location Awareness in UbiCollab
described. Last in this chapter the implementation of the UbiHome server will be described. The UbiBuddy application will be described in the next chapter.

All of the platform components are implemented as OSGi bundles, running on the Knopflerfish framework. The API of these bundles is accessible through Web Service interfaces. By doing this, the platform components become independent of each other and the OSGi framework. The applications built on top of the platform also become independent of programming language. The Collaboration Space Manager Tool and the Collaboration Instance Manager Tool are implemented in the Microsoft Visual Studio .Net framework. This proves the concepts of programming language independency. The Collaboration Instance updater tool(CIUpdaterTool) is developed as a bundle in the OSGi framework. This bundle does not provide any API, but uses the API provided by the other components in the platform to publish data in the Collaboration Instance Manager.

6.2 Collaboration Space Manager

The Collaboration Space Manager handles the management of Collaboration Spaces. This platform component provides the necessary functionality to all the parts of UbiCollab. By providing the coordinates of a physical location, the Collaboration Space Manager finds the matching Collaboration Space. The bundle also has functionality to add, remove and modify Collaboration Spaces.

A Collaboration Space is in our implementation represented with the following data:
- Name
- Latitude
- Longitude
- Radius
- Granularity
- Parent
- Properties

The name describes the Collaboration Space. The latitude and longitude are used to define where the Collaboration Space is located. We have made a simplification with regards to the shape of Collaboration Spaces; we can only define them as circles, therefore we use the radius in order to define the size of the Collaboration Space. The granularity level is to define the level of abstraction for the Collaboration Space. In order to represent Collaboration Spaces in a hieratical structure, a parent Collaboration Space can be defined. The property field is used to describe special properties of the location. For instance one can add “mobile” as a property if the Collaboration Space is a car.

6.2.1 API

The API for this bundle is shown in Figure 6.2. This interface provides the functionality that is expected from the Collaboration Space Manager. This API is accessible
through a Web Service interface. Following is a description of the functionality provided by this interface.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getCsString()</code></td>
<td>Returns a String, containing all the Collaboration Spaces.</td>
</tr>
<tr>
<td><code>login()</code></td>
<td>This method connects to the UbiHome Server, and retrieves the Collaboration Spaces of the user. This enables the user to use the same set of Collaboration Spaces wherever he or she logs in.</td>
</tr>
<tr>
<td><code>setCS()</code></td>
<td>Used to set the user's current Collaboration Space.</td>
</tr>
<tr>
<td><code>addCS()</code></td>
<td>Used to create new Collaboration Spaces.</td>
</tr>
<tr>
<td><code>modifyCS()</code></td>
<td>Used to modify existing Collaboration Spaces.</td>
</tr>
<tr>
<td><code>deleteCS()</code></td>
<td>Used to delete Collaboration Spaces</td>
</tr>
<tr>
<td><code>getCurrentCS()</code></td>
<td>Used to retrieve the current Collaboration Space</td>
</tr>
<tr>
<td><code>getCSRec()</code></td>
<td>Returns the recommended Collaboration Space based on your current position. This requires GPS coordinates.</td>
</tr>
</tbody>
</table>

Figure 6.2: Collaboration Space Manager API

6.2.2 Class diagram

Figure 6.3 shows the class diagram of the Collaboration Space Manager. The class `no.ubicollab.osgi.CSManager` is the Web Service interface provided by the Collaboration Space Manager. The functionality behind this interface is provided by the `no.ubicollab.osgi.CSManagerImpl` class. This class has a list of CS objects in order to manage the different Collaboration Spaces of each user.

6.3 csmTool

The csmTool is used to alter data in the Collaboration Space Manager. This is done by using the API specified in Figure 6.2. A screenshot of this tool is shown in Figure 6.4. The Tool is developed in the Microsoft Visual Studio .Net framework, using C#. To alter data in the Collaboration Instance Manager, the different applications need to access this tool. It's a standalone implementation, available for the user, either from the application, or by starting this directly.
6.4 Collaboration Instance Manager

The Collaboration Instance Manager is implemented as an OSGi bundle, running on the Knopflerfish framework. This element provides community management functionality needed at the platform level. This bundle has the functionality to create, delete and modify Collaboration Instances. Users can be connected to different Collaboration Instances, and has the ability to publish data, making the data available for other participants.

6.4.1 API

The API for this bundle is shown in Figure 6.3. The API is accessible through a Web Service interface.

- `login()` Retrieves the users Collaboration Instances from the UbiHome server.
- `getCollaborationInstanceID()` Returns Collaboration Instance id’s as a String.
- `getCollaborationInstanceMembers()` Returns the members of a Collaboration Instance.

Espen Larsen Segelvik and Morten Larsen Segelvik
• `getCollaborationInstanceOwner()` - Returns True if the user is owner of the Collaboration Instance.

• `getCollaborationInstanceName()` - Returns the name of the Collaboration Instance.

• `getCurrentCS()` - Retrieves the current Collaboration Space of a user published in the Collaboration Instance.

• `createCI()` - Used to create a Collaboration Instance.

• `addUserToCI()` - Used to add users to the Collaboration Instance.

• `deleteUserFromCI()` - Used to delete users from a Collaboration Instance.

• `deleteCI()` - Used to delete Collaboration Instances.

• `setUserUrl()` - Updates the user’s current URL.

• `getUserUrl()` - Returns the current URL of a given user.

• `updateCIGranularity()` - Used to update the granularity level in the different Collaboration Instances.

• `getCIGranularity()` - Returns the granularity level for the user in a given Collaboration Instance.

• `getCIList()` - Returns the CI list objects.

• `setCIList()` - Used to update the CI list.
• **getUserServices()**: Returns the list of services published in the Collaboration Instance by a user.

![CIManager](image)

**Figure 6.5: Collaboration Instance Manager API**

### 6.4.2 Class diagram

A class diagram for the Collaboration Instance Manager is shown in Figure 6.6. The interface no.ubicollab.CIManager is implemented as a Web Service interface and the functionality behind is implemented in no.ubicollab.org.CIManagerImpl. This arranges the different Collaboration Instances in CI objects. Each CI object has a list of users and services.

### 6.5 cimTool

The cimTool is used to alter data in the Collaboration Instance Manager. This is done by using the API specified in Figure 6.5. A screenshot of this tool is shown in Figure 6.7. The tool is developed in the Microsoft Visual Studio .Net framework, using C#. This shows the flexibility of the platform developed in Java. By utilizing the API provided by the Collaboration Space Manager, this tool provides a graphical user interface for the management of communities in UbiCollab.

### 6.6 CIUpdaterTool

As described in the previous chapter, the interface to application developers is the Collaboration Instance Manager. Thus, showing information about Collaboration Spaces in the applications requires a tool to publish data in the Collaboration Instances. We have implemented a CIUpdaterTool which gather information about the current Collaboration Space of users and publish this information in the Collaboration Instance Manager. This tool can also provide the Collaboration Space...
Manager with coordinates from a GPS service, which enables the user to automatically change the Collaboration Space based on his or her current position.

This tool uses the Service Domain Manager, implemented by another master student. The tool updates information about services published in Collaboration In-
stances. When the services change, the tool updates the services published in Collaboration Instances. Another task for this tool with regards to the Service Domain Manager is to activate and deactivate services of a user, based on the current Collaboration Space of the user.

### 6.6.1 Class diagram

The class diagram of the CIUpdaterTool is shown in Figure 6.8. This component has a thread called Updater, which manages the publishing of information in Collaboration Instances, as well as keeping the Collaboration Instance data synchronized. This can be seen as a connection between the Collaboration Space Manager, Service Domain Manager and the Collaboration Instance Manager. To keep the Collaboration Instance Manager updated with the correct information, it sends a request to the different users in order to retrieve information in regular intervals.

![Figure 6.8: Collaboration Instance Updater Tool Class Diagram](image.png)

### 6.7 UbiHome

The UbiHome server was not designed in the previous chapter, due to the fact that this is out of the scope of this thesis. However we have developed a version of the UbiHome server in order to have a working demonstrator. The UbiHome server is implemented as a Web Service interface. The interface is used to store, retrieve and modify data used in UbiCollab. The UbiHome server provides a persistent data storage in a MySQL database. The following interface is implemented:

- **addCS()** Creates a new Collaboration Space in the database.
• **addUser()** Creates a new user in the database.
• **createCI()** Creates a Collaboration Instance in the database.
• **deleteCI()** Deletes a Collaboration Instance from the database.
• **getAll()** Returns a xml file with all the information needed to initialize a Collaboration Instance Manager.
• **getCS()** Returns information about a given Collaboration Space.
• **joinCI()** Adds a user to a Collaboration Instance.
• **leaveCI()** Removes a user form a Collaboration Instance.
• **modifyCS()** Modifies a Collaboration Space.
• **removeCS()** Deletes a Collaboration Space.
• **setUrl()** Updates the current URL of a user.
• **getPublishedServices()** Returns the services published to a Collaboration Instance.
• **getPublishedServicesUser()** Returns the services published by a user.
• **publishService()** Publishes a service in a Collaboration Instance.
• **unpublishService()** Unpublishes a service in a Collaboration Instance.

6.8 Implementation vs. Design

The implementation differs from the design in some places. The implemented differences are done in order to ease the implementation. We will in this section go through the differences in each component implemented, starting with the Collaboration Space Manager. This is followed by the differences in the Collaboration Instance Manager.

6.8.1 Collaboration Space Manager

Table 6.1 shows the designed API for the Collaboration Space Manager, and where they are covered by the implemented API. As we see from the Table, the API implemented only has one naming difference. This does not affect the functionality provided. The functionality of this component and the fulfillment of functional requirements will be discussed in the next chapter.

<table>
<thead>
<tr>
<th>Design API</th>
<th>Implementation API</th>
</tr>
</thead>
<tbody>
<tr>
<td>login</td>
<td>login</td>
</tr>
<tr>
<td>getCSData</td>
<td>getCSString</td>
</tr>
<tr>
<td>addCS</td>
<td>addCS</td>
</tr>
<tr>
<td>deleteCS</td>
<td>deleteCS</td>
</tr>
<tr>
<td>modifyCS</td>
<td>modifyCS</td>
</tr>
</tbody>
</table>

Continued on next page
### 6.8. Implementation vs. Design

#### Table 6.1 – continued from previous page

<table>
<thead>
<tr>
<th>Design API</th>
<th>Implementation API</th>
</tr>
</thead>
<tbody>
<tr>
<td>getCurrentCS</td>
<td>getCurrentCS</td>
</tr>
<tr>
<td>setCurrentCS</td>
<td>setCS</td>
</tr>
<tr>
<td>getNearestCS</td>
<td>getCSRec</td>
</tr>
</tbody>
</table>

Table 6.1: Design API vs. Implementation API - Collaboration Space Manager

#### 6.8.2 Collaboration Instance Manager

In Table 6.2 the difference from the designed API and the implementation API for the Collaboration Instance Manager are shown. The biggest difference is the division of the getCIData method designed. As well as providing this method in the getCIList method, we provide a more detailed API. All the smaller methods provide an interface to extract only the information an application requires. This is feasible due to the flexibility this approach introduces. In the original design, to get small amounts of information, the application had to ask for everything. This is avoided by introducing the more detailed methods in the interface. These changes do not affect the functionality of the Collaboration Instance Manager, just the interface.

A difference that affects the functionality is the publishData and getPublishData interface. The publishData is more or less done through the CIPusherTool. The implementation of this component is described in Section 6.6. The implementation of the Collaboration Instance Manager does not provide a generic interface for the publishing of data. The CIPusherTool uses the getCIList interface in order to retrieve the complete list of Collaboration Instances. The tool then retrieves data about Collaboration Spaces and services. The tool publishes data by writing the new and updated list of Collaboration Instances back to the Collaboration Instance Manager by using the setCIList interface.

<table>
<thead>
<tr>
<th>Design API</th>
<th>Implementation API</th>
</tr>
</thead>
<tbody>
<tr>
<td>login</td>
<td>login</td>
</tr>
<tr>
<td>getCIData</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• getCollaborationInstanceID</td>
</tr>
<tr>
<td></td>
<td>• getCollaborationInstanceMembers</td>
</tr>
<tr>
<td></td>
<td>• getCollaborationInstanceOwner</td>
</tr>
<tr>
<td></td>
<td>• getCollaborationInstanceName</td>
</tr>
<tr>
<td></td>
<td>• getUserUrl</td>
</tr>
<tr>
<td></td>
<td>• getUserServices</td>
</tr>
<tr>
<td></td>
<td>• getCIGranularity</td>
</tr>
<tr>
<td></td>
<td>• getCIList</td>
</tr>
<tr>
<td>createCI</td>
<td>createCI</td>
</tr>
<tr>
<td>deleteCI</td>
<td>deleteCI</td>
</tr>
<tr>
<td>modifyCI</td>
<td>updateCIGranularity</td>
</tr>
<tr>
<td>addUser</td>
<td>addUserToCI</td>
</tr>
<tr>
<td>deleteUser</td>
<td>deleteUserFromCI</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 6.2: Design API vs. Implementation API - Collaboration Instance Manager

<table>
<thead>
<tr>
<th>Design API</th>
<th>Implementation API</th>
</tr>
</thead>
<tbody>
<tr>
<td>publishData</td>
<td>setCIList</td>
</tr>
<tr>
<td>getPublishedData</td>
<td>• getUserServices</td>
</tr>
<tr>
<td></td>
<td>• getCIList</td>
</tr>
<tr>
<td>Not in design</td>
<td>setCIList</td>
</tr>
</tbody>
</table>
Chapter 7

Demonstrator

*The important achievement of Apollo was demonstrating that humanity is not forever chained to this planet and our visions go rather further than that and our opportunities are unlimited*

-Neil Armstrong
In this chapter, we will present a working implementation of a demonstrator. We start with a brief introduction to the demonstrator, followed by a technical description. Then we will go through the same scenario described in Chapter 2. This will illustrate the functionality at the application level. Then a description of the actions at the platform level is given. Finally we evaluate our design and implementation with regards to functional and non-functional requirements.

### 7.1 Application - UbiBuddy

We have developed an application called UbiBuddy. This is an enhanced buddy list built on top of the UbiCollab platform, with specific focus on the collaborative support provided by UbiCollab. The application accesses data stored in the Collaboration Instance Manager which provides the API for the application. To alter data in the platform, UbiCollab uses a set of tools provided by the platform. These tools are standalone implementations which are launched from UbiBuddy.

The prototype shows how awareness information can be collected from and distributed into different collaboration spaces using physical tools and artifacts. UbiBuddy overcomes shortcomings of available buddy list applications by making the application ubiquitous and community-aware. UbiCollab provides support for physical interfaces as well as mobility.

Different communities need different information. By using the UbiCollab platform, information can be tailored to each community. In ordinary buddy list applications, all communities receive the same information. This information could for instance be information about their physical location and availability.

Mobility in normal buddy list applications is restricted to accessing the application from mobile devices. In UbiCollab users can define their own Collaboration Spaces and configure devices within these. This enables the user to access different devices based on the user location, and UbiBuddy can utilize this in order to customize information based on location.

### 7.2 UbiBuddy implementation

We have implemented a version of UbiBuddy which is built on top of the UbiCollab platform. It communicates with the platform through a Web Service interface provided by the Collaboration Instance Manager. To demonstrate the independency from the platform we developed the application in the Microsoft Visual Studio .Net framework using information from OSGi bundles implemented in Java. Applications are basically a GUI which utilizes information provided by the Collaboration Instance Manager. When the user logs in to UbiBuddy he or she gets a list of all Collaboration Instances the user is a member of. When a user starts UbiBuddy, a list of Collaboration Instances and the members of these are shown. By clicking on a user, information about published services is shown. Since this information is very dynamic, and data in the Collaboration Instance change often, we choose to implement a thread pulling for information. The functionality of this application is illustrated in the next section.
7.3 Demonstrator Scenario

In the following we report the demonstration scenario and illustrates the actions with screenshots from our implementation described in the Chapter 6.

**Action 1:** Pedro is using UbiBuddy to stay in touch with his family and friends. UbiBuddy is on his Tablet PC which he usually brings wherever he goes.

A picture of Pedro and his Tablet PC are shown in Figure 7.1.

![Figure 7.1: Pedro using UbiBuddy](image)

**Action 2:** He currently has set up two communities, friends and family. He has also defined a Collaboration Space connected to his office.

The two communities are shown in UbiBuddy in Figure 7.2.

**Action 3:** He has just met a group of environmental activists and he decides to use UbiBuddy to stay in touch with them, so he creates a new community and he adds Jorge and Lola. He will ask later to the others to register to UbiCollab, so that he can add them.

The creation of a Collaboration Instance is shown in Figure 7.3 and the adding of users in Figure 7.4.

---

Espen Larsen Segelvik and Morten Larsen Segelvik
Action 4: Pedro can now see the new community in his list of communities with the registered participants.

This is shown in Figure 7.5.

Action 5: He decides to set his status as not available to the new community when he is at school.

Action 6: Pedro has just moved into his new apartment and he wants to configure the devices that he has available so that he can use them easily to stay in touch.

For this he creates a new Collaboration Space, "Home". This is shown in Figure 7.6.

Location Awareness in UbiCollab
7.3. Demonstrator Scenario

Action 7: He has just bought a lamp and he decides to put it in the corridor and to make it available to his friends as an awareness display.

This is shown in Figure 7.7.

Action 8: Birgit, one of Pedro’s friends, connects to UbiBuddy and she sees that Pedro is at home and he has made available the cool lamp she has suggested him to buy. She clicks on the lamp icon twice to send him a quick message. Maybe he will get curious and check who is on-line.

Action 9: Pedro actually gets curious of who is turning on his lamp, so he checks who is on-line by looking at his tablet. He sees that Birgit is on-line and he calls Espen Larsen Segelvik and Morten Larsen Segelvik.
her on the VOIP tool that she has made available.

**Action 10:** It is now time to go to work. When Pedro arrives at his office Ubibuddy adapts to his new location. The main screen in his office becomes now the main display for visualizing awareness about his contacts.

**Action 11:** Pedro is in his office. When there he has made available his coffee machine as a service to his friends. They like that they can use UbiBuddy to check when he is in his office and that they can turn on the coffee machine before popping by for a break.

By double clicking on a user in UbiBuddy, a new window with pops up with the
services shared. Figure 7.8 shows services active/inactive when he is at home. While Figure 7.9 shows services active/inactive when he is at the office.

![Figure 7.8: Services available at home](image)

(a) Active Home  
(b) Inactive Home

![Figure 7.9: Services available at office](image)

(a) Active Office  
(b) Inactive Office

**Action 12:** Pedro is traveling to Spain and he has no equipment with him. He borrows a mobile phone from his cousin. The phone has the UbiCollab platform installed and he can easily get all his settings.

### 7.4 Action at the platform level

The scenario described in the previous section, only shows what happens at the application level. Table 7.1 shows what happens at the platform level for each of the actions in the scenario.

<table>
<thead>
<tr>
<th>Scenario action</th>
<th>Action at the platform level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action 1</td>
<td>UbiCollab is running on the platform with at least the Collaboration Instance Manager installed.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 7.1 – continued from previous page

<table>
<thead>
<tr>
<th>Scenario action</th>
<th>Action at the platform level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action 2</td>
<td>Same as action 1.</td>
</tr>
<tr>
<td>Action 3</td>
<td>The Collaboration Instance Manager receives a addCollaborationInstance call. The Collaboration Instance is created in the Collaboration Instance Manager and at the UbiHome server. Then the Collaboration Instance Manager receives two addUserToCI calls. The users are added in the Collaboration Instance Manager and at the UbiHome server.</td>
</tr>
<tr>
<td>Action 4</td>
<td>Same as action 1.</td>
</tr>
<tr>
<td>Action 5</td>
<td>Not supported by the current version of UbiCollab.</td>
</tr>
<tr>
<td>Action 6</td>
<td>Pedro starts the csmTool and creates a new Collaboration Space. This is added to the list of Collaboration Spaces in the Collaboration Space Manager, which in turn updates the UbiHome server.</td>
</tr>
<tr>
<td>Action 7</td>
<td>To make the lamp available for his friends, he has to publish this service in a Collaboration Instance with the Service Management tool. The Collaboration Instance is updated in the Collaboration Instance Manager and at the UbiHome server. The service is now made available for the users in that Collaboration Instance.</td>
</tr>
<tr>
<td>Action 8</td>
<td>Partly supported in the current version. When double clicking on a person, UbiBuddy sends a query to the Collaboration Instance Managers about the services this user has published in the selected Collaboration Space. However it has not been developed any means of controlling these services.</td>
</tr>
<tr>
<td>Action 9</td>
<td>Not supported by the current version of UbiCollab.</td>
</tr>
<tr>
<td>Action 10</td>
<td>Pedro has a tool that automatically retrieves his position from a location device, and updates the current Collaboration Space by requesting this from the Collaboration Space Manager. This tool also updates services available/unavailable through the API provided by the Service Domain Manager.</td>
</tr>
<tr>
<td>Action 11</td>
<td>Partly supported in the current version. When double clicking on a person, UbiBuddy sends a query to the Collaboration Instance Managers about the services this user has published in the selected Collaboration Space. However it has not been developed any means of controlling these services.</td>
</tr>
<tr>
<td>Action 12</td>
<td>When a user logs in on another UbiCollab enabled device, his personal settings are downloaded from the UbiHome server, and the Collaboration Space Manager and Collaboration Instance Manager is working as normal.</td>
</tr>
</tbody>
</table>

Table 7.1: Actions at the platform level

7.5 Evaluation of the results and limitations

In this section we will evaluate the design, implementation and the demonstrator of the components we have focused on. We will start with looking into how the requirements are fulfilled, first the functional, followed by the non-functional requirements. Finally we look into limitations of our implementation unveiled by the demonstrator.
7.5. Evaluation of the results and limitations

7.5.1 Functional requirements

Table 7.2 shows which of the functional requirements that are fulfilled. Most of the functional requirements are fulfilled by our work, except for CS-FREQ4 which is fulfilled by another master student[12]. There are made some simplifications in CI-FREQ5 due to limited resources and the fact that the Collaboration Instance Manager was not the main focus of our work. The simplifications was described in Section 6.8.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
<th>Fulfilment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS-FREQ1</td>
<td>Create Collaboration Space</td>
<td>Fulfilled</td>
</tr>
<tr>
<td>CS-FREQ2</td>
<td>Delete Collaboration Space</td>
<td>Fulfilled</td>
</tr>
<tr>
<td>CS-FREQ3</td>
<td>Modify Collaboration Space</td>
<td>Fulfilled</td>
</tr>
<tr>
<td>CS-FREQ4</td>
<td>Connect Services to Collaboration Space</td>
<td>Not our task, but fulfilled by [12]</td>
</tr>
<tr>
<td>CS-FREQ5</td>
<td>Provide an interface in order to retrieve Collaboration Space information</td>
<td>Fulfilled</td>
</tr>
<tr>
<td>CS-FREQ6</td>
<td>Provide functionality in order to retrieve the current Collaboration Space of a user</td>
<td>Fulfilled</td>
</tr>
<tr>
<td>CS-FREQ7</td>
<td>Automatic update of Collaboration Space based on location</td>
<td>Fulfilled</td>
</tr>
<tr>
<td>CI-FREQ1</td>
<td>Add Collaboration Instance</td>
<td>Fulfilled</td>
</tr>
<tr>
<td>CI-FREQ2</td>
<td>Delete Collaboration Instance</td>
<td>Fulfilled</td>
</tr>
<tr>
<td>CI-FREQ3</td>
<td>Modify Collaboration Instance</td>
<td>Fulfilled</td>
</tr>
<tr>
<td>CI-FREQ4</td>
<td>Associate users to a Collaboration Instance</td>
<td>Fulfilled</td>
</tr>
<tr>
<td>CI-FREQ5</td>
<td>Share data in Collaboration Instance</td>
<td>Partly fulfilled</td>
</tr>
</tbody>
</table>

Table 7.2: Fulfillment of functional requirements

7.5.2 Non-functional requirements

As shown in Table 7.3 most of the non-functional requirements are fulfilled within our design and implementation.

Regarding scalability, we have tried to keep the communication between nodes at a minimum. However this has not been tested with a large number of nodes, and it is hard to evaluate the fulfillment of this requirement.

Our implementation is developed in Java, thus it should be able to run on any device running the java virtual machine. Thus the requirement regarding OS independency is considered fulfilled.

The service oriented architecture and a fixed API provided to the application developers have shown flexible to use in the demonstrator. However it is hard to foresee every application that will be built on top of UbiCollab, thus fulfillment of the requirement regarding end user programming is hard to evaluate.

Each component in UbiCollab has a fixed responsibility, thus the platform should be easy to extend. Tools can be implemented to combine resources from different

Espen Larsen Segelvik and Morten Larsen Segelvik
components, and provide new functionality. Thus this requirement is considered fulfilled.

As we have shown thorough the design and implementation, each component is developed based on SOA principles. The components can be modified to accommodate changes in technology, and will not affect the functionality provided as long as the API remains constant. Thus this requirement is fulfilled.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
<th>Fulfilment</th>
</tr>
</thead>
<tbody>
<tr>
<td>N-FREQ1</td>
<td>Scalability</td>
<td>Unknown</td>
</tr>
<tr>
<td>N-FREQ2</td>
<td>OS independency</td>
<td>Fulfilled</td>
</tr>
<tr>
<td>N-FREQ3</td>
<td>End user programming</td>
<td>Unknown</td>
</tr>
<tr>
<td>N-FREQ4</td>
<td>Extendibility</td>
<td>Fulfilled</td>
</tr>
<tr>
<td>N-FREQ5</td>
<td>Service oriented architecure</td>
<td>Fulfilled</td>
</tr>
</tbody>
</table>

Table 7.3: Fulfillment of non-functional requirements

7.5.3 Limitations

In this section we will describe the limitations of our implementation. We start with the Collaboration Space Manager, followed by the Collaboration Instance Manager.

Collaboration Space Manager

In our implementation there are some limitations with respect to the Collaboration Space Manager. The most obvious limitation is that all Collaboration Spaces are defined as circles. A Collaboration Space could be any geometric shape. This is not supported in the current version of the Collaboration Space Manager. We leave this for the future work to implement a generic way of defining the shape of a Collaboration Space. For demonstrator purposes the circle implementation works fine.

Regarding the Collaboration Space Manager tool, we feel that this should provide the functionality to automatically retrieve GPS coordinates from a positioning device. In the current implementation, the users must manually obtain these when managing the Collaboration Spaces.

Collaboration Instance Manager

In the implementation of the Collaboration Instance Manager, we have a limitation regarding the publishing of data in a Collaboration Instance. We only publish Collaboration Spaces and services in our implementation, but it should be extended to support other data relevant to the Collaboration Instance. As mentioned in the design chapter, this method should be generic. We suggest that this method require three variables: Collaboration Instance ID, name and value. In our implementation we have two static interfaces for publishing Collaboration Space and services. We leave that for future work to implement the generic interface for the publishing of collaborative relevant data.
A second limitation is the synchronization of Collaboration Instances. The Collaboration Instance contains much dynamic and distributed information which must be synchronized among all nodes in UbiCollab. We have not done any elaboration of how this synchronization should be done. The solution we used was the implementation of the CIUpdaterTool described in Section 6.6. We leave the synchronization for future work.
7. Demonstrator
Chapter 8

Conclusion

*Reasoning draws a conclusion, but does not make the conclusion certain, unless the mind discovers it by the path of experience.*

-Roger Bacon
8. Conclusion

8.1 Summary

UbiCollab is a platform that aims to aid collaboration on the internet in ubiquitous environments. Our task was focused on location and community awareness in UbiCollab. One key aspect when designing location-aware system is taking into account that the users are members of different communities, and locations have different meanings in these. We started with a problem elaboration to identify high level needs for our work within UbiCollab. To get a more in-depth understanding of the problem and the problem domain, we conducted a literature study. This led to the platform requirements that our design should fulfill. From the requirements we designed and implemented a component, the Collaboration Space Manager, which provides location as a resource for collaboration. To show the diversity in our implementation, we designed and implemented the Collaboration Instance Manager. This is the component responsible for the management of communities in UbiCollab. As we have shown throughout the report, it is endless possibilities connected to location. Locations of users can help user become aware of each others context, services available can be customized based on location etc. We have proposed a design which is technology independent of positioning devices used. By integrating the Collaboration Space Manager with the component responsible for managing services in UbiCollab, we have shown that our work can be utilized as a resource for collaboration. To demonstrate strengths and weaknesses in our design and implementation of these two components, we have implemented an application, UbiBuddy. This is a ubiquitous buddy list running on top of UbiCollab. In the demonstrator the Collaboration Space Manager is utilized in order to provide awareness of the different users, and management of services used for collaboration.

8.2 Contributions

Throughout this thesis we have contributed in various ways in our research area. Our main goal was to design, implement, and evaluate UbiCollab components to support the usage of location information as a resource for collaboration in distributed and ubiquitous settings. We started with a problem elaboration, where we presented a scenario illustrating the behavior of UbiCollab and UbiBuddy. This contributed to a more thorough understanding of the research field, and we used this to formulate the following subgoals:

- **Subgoal 1**: Identification of requirements at the platform level to make location information available as a resource for collaboration in UbiCollab.
- **Subgoal 2**: Define and implement a location model that fulfills UbiCollab requirements and the need to associate different descriptions to places.
- **Subgoal 3**: Design and prototype mechanisms for supporting the sharing of location information within communities.
- **Subgoal 4**: Design and prototype mechanisms to use location information to support the service management.
- **Subgoal 5**: Implement a demonstrator showing the strengths and weaknesses of our design and implementation.
8.3. Further work

To fulfill subgoal 1, we started with a literature study, giving us an understanding of location awareness in general, positioning technologies and different location models for storing and representing location in location-aware systems. From the problem elaboration and literature study we identified requirements at the platform level to make location information available as a resource for collaboration in UbiCollab.

Based on these results, we moved on to subgoal 2, where we defined and implemented a location model that fulfilled UbiCollab requirements. Most of the location-aware systems today rely on a centralized solution for managing locations. We found this solution inadequate, due to the fact that locations have different meanings for different individuals. As described in Chapter 3, Place Lab is a system where the users creates and maintain the location model. Our work differs from this by letting each user have their own location model that they are responsible for maintaining. This led to the implementation of a user centered Collaboration Space Manager. As illustrated in this thesis, this is a very flexible way of maintaining locations in UbiCollab. Since the association of locations is dependent on the users residing in them, each user can create their own personal model.

Subgoal 3 was fulfilled by the design and implementation of the Collaboration Instance Manager. This is the component managing communities and sharing of information among them. Due to limited resources, this was a very simple design and implementation. The implementation of this component support basic community management and sharing of Collaboration Spaces and services.

Subgoal 4 is based on the results from subgoals 2 and 3. We integrated our solution with the Service Management part of the platform[12]. Based on our work, the user has the ability to share services in different communities, and automatically configure these based on location.

We have shown the strengths and weaknesses of our design and implementation by implementing a demonstrator, which where subgoal 5. The demonstrator shows how location can be used in different ways regarding collaboration on the internet. In the demonstrator, UbiBuddy, the location was both used for achieving awareness of the users, and for the management of services. In our work services automatically becomes available/unavailable based on the location of the users. This is further tailored to each Collaboration Instance. However there were some limitations in our design and implementation. These limitations are described in the Demonstrator chapter, in Section 7.5.

8.3 Further work

Regarding our implementation, there are several improvements that should be considered in future development of UbiCollab. In this section we will describe improvements to the Collaboration Space Manager and the Collaboration Instance Manager.

8.3.1 Collaboration Space Manager

During the work of this thesis, we identified some trouble with the notion Collaboration Space. As for now, the Collaboration Space Manager does not distinguish
between the space that you have in your personal model, and the one you share with other collaborating users. When using spaces for collaborating, by sharing the space with appurtenant artifacts, one might want to remove or add some properties, making the shared space somewhat different from the one you have in your personal model. We suggest dividing the Collaboration Space Manager in two parts; a Space Manager and a Collaboration Space Manager. Where the Space Manager is the personal model of a user, and the Collaboration Space Manager is the space shared among collaborating users. During this thesis, Professor Dr. Monica Divitini and Dr. Babak A. Farschian have been writing Architecture White Paper for UbiCollab. Our feedback regarding these concepts has been accounted for in the latest version. The latest version of the UbiCollab Architecture White Paper is attached in Appendix [3].

One obvious improvement to the Collaboration Space Manager is the support for shapes. In the current implementation we only support the definition of circles. It is obvious that locations can have various shapes, and we suggest that the next implementation iteration support various shapes.

Another limitation to this component is the calculation of Collaboration Space, based on coordinates. In the current implementation only GPS coordinates are used. In the future, with new and emerging technologies, this might not become the standard, and it might be useful to take into consideration the support of different coordinate systems. However, this was not the main focus of our task, thus we have not implemented a solution for this.

Altitude has not been taken into account. Due to the fact that the altitude coordinate provided from the GPS system is very inaccurate, it would be useless for us to implement support for this. In large buildings altitude is required in order to determine the correct floor. For future development we suggest that the Collaboration Space Manager is extended in order to take into account the altitude of a user. The fact that we uses GPS devices to retrieve the location of a user, limits the usability of this implementation. The GPS signal retrieved inside is often poor. Therefore we suggest that one should use more accurate sources for indoor positioning.

### 8.3.2 Collaboration Instance Manager

The main issue that needs an improvement is the interface for publishing data. As it is now, a tool retrieves the whole list of Collaboration Instances and publishes data, and updates the whole list of Collaboration Instances. Another issue with regards to publishing data is that it is not general enough. The only data the Collaboration Instance Manager supports is services and Collaboration Spaces. An improvement would be to create a generic interface in order to enable users to publish all relevant data. We did not see this part of the Collaboration Instance Manager as an important issue within our task, and we leave this for future development.

Another important issue regarding the Collaboration Instance Manager is the synchronizing of Collaboration Instances. As for the current implementation, the synchronization is done through the same tool that publishes data in a Collaboration Instance. The tool requests other users Collaboration Instance Managers, and retrieves information shared within them. This is not a very elegant method to use, but it works and proves the Collaboration Instance concept. This is outside the scope of our task, and we leave this for future work.
Bibliography

[1] Place lab. http://www.placelab.org/[Last accessed 01.05.2007].


Location Awareness in UbiCollab


Location Awareness in UbiCollab
Appendix A

Installation Instructions
A.1 Required Software

The Collaboration Instance Manager, Collaboration Space Manager and the CIUpdaterTool are developed as OSGi bundles. In order to run OSGi bundles the following software are required:

- **Java 5.0**
  - URL: [http://java.sun.com/j2se/corejava/index.jsp](http://java.sun.com/j2se/corejava/index.jsp)

- **Knopflerfish OSGi framework**

UbiBuddy, Collaboration Instance Manager Tool and Collaboration Space Manager Tool are developed in the .Net framework. The software required for these are:

- **Microsoft .NET Framework Version 2.0**

The UbiHome server is a Web Service deployed on a Microsoft web server. It uses a MYSQL database to store data. The software required for UbiHome is:

- **MySQL Database**
  - URL: [http://dev.mysql.com/downloads/mysql/5.0.html](http://dev.mysql.com/downloads/mysql/5.0.html)

A.2 Quick setup guide

1. Install the neccesary software described above.
2. Launch the Knoplerfish OSGi framework.
3. Load CIManager, CSManager and DomainManager in Knopflerfish.
4. Log into these bundles either through the csmTool and cimTool, or the login application.
5. Load UbiBuddy.
6. The CIUpdaterTool can be launched; this requires the Service Domain Manager.

A.3 Binaries and Source

Table A.1 shows the path to the binaries and source relative to the structure of the zip file.

<table>
<thead>
<tr>
<th>Name</th>
<th>Binaries</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIManager</td>
<td>masterthesis/bin/bundles/CIManager-1.0.0.jar</td>
<td>masterthesis/src/bundles/CIManager</td>
</tr>
<tr>
<td>CSManager</td>
<td>masterthesis/bin/bundles/CSManager-1.0.0.jar</td>
<td>masterthesis/src/bundles/CSManager</td>
</tr>
<tr>
<td>CIUpdaterTool</td>
<td>masterthesis/bin/bundles/CIUpdaterTool-1.0.0.jar</td>
<td>masterthesis/src/bundles/CIUpdaterTool</td>
</tr>
<tr>
<td>cimTool</td>
<td>masterthesis/bin/net/cimTool.exe</td>
<td>masterthesis/src/net/cimTool</td>
</tr>
</tbody>
</table>

Continued on next page
Table A.1 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Binaries</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>csmTool</td>
<td>masterthesis/bin/net/csmTool.exe</td>
<td>masterthesis/src/net/csmTool</td>
</tr>
<tr>
<td>LogInApp</td>
<td>masterthesis/bin/net/LogInApp.exe</td>
<td>masterthesis/src/net/LogInApp</td>
</tr>
<tr>
<td>UbiBuddy</td>
<td>masterthesis/bin/net/UbiBuddy.exe</td>
<td>masterthesis/src/net/UbiBuddy</td>
</tr>
<tr>
<td>UbiHome</td>
<td></td>
<td>masterthesis/src/net/UbiHome</td>
</tr>
<tr>
<td>GPSProxyBundle</td>
<td>masterthesis/bin/bundles/__GPProxyBundle-1.0.0.jar</td>
<td>masterthesis/src/bundles/__GPProxyBundle</td>
</tr>
<tr>
<td>DomainManager</td>
<td>masterthesis/bin/bundles/__domainmanager.jar</td>
<td>Not developed by us</td>
</tr>
</tbody>
</table>

Table A.1: Implementation Attachments

## A.4 UbiHome Database

Listing A.1 shows the database setup required in order for launching the UbiHome server in a proper way.

```sql
DROP TABLE IF EXISTS 'ubihome'.'collaborationinstance';
CREATE TABLE 'ubihome'.'collaborationinstance' (  
   'CIID' int(11) NOT NULL auto_increment,  
   'CIName' varchar(40) NOT NULL,  
   'CIData' text,  
   PRIMARY KEY ('CIID')  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS 'ubihome'.'collaborationinstance_user';
CREATE TABLE 'ubihome'.'collaborationinstance_user' (  
   'CIID' int(11) NOT NULL,  
   'Username' varchar(40) NOT NULL,  
   'Owner' tinyint(1) NOT NULL,  
   'Granularity' varchar(40) default NULL,  
   PRIMARY KEY ('CIID', 'Username'),  
   KEY 'User_CollaborationInstance_User' ('Username'),  
   CONSTRAINT 'CollaborationInstance_CollaborationInstance_User' FOREIGN KEY ('Username') REFERENCES 'collaborationinstance' ('CIID'),  
   CONSTRAINT 'User_CollaborationInstance_User' FOREIGN KEY ('Username') REFERENCES 'user' ('Username')  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS 'ubihome'.'published_services';
CREATE TABLE 'ubihome'.'published_services' (  
   'CIID' varchar(45) NOT NULL,  
   'SID' varchar(45) NOT NULL,  
   'username' varchar(45) NOT NULL,  
   PRIMARY KEY ('CIID', 'SID'),  
   KEY 'PublishedServices_CIID' ('CIID'),  
   CONSTRAINT 'PublishedServices_CollaborationInstance' FOREIGN KEY ('CIID') REFERENCES 'collaborationinstance' ('CIID')  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Espen Larsen Segelvik and Morten Larsen Segelvik
Listing A.1: Create statements for the UbiHome MySQL database
1 What is UbiCollab? ........................................................................................................... 2
2 UbiCollab Basic Concept: Human Grid........................................................................... 2
3 UbiCollab Example Scenario............................................................................................ 3
4 UbiCollab design philosophy ............................................................................................ 5
5 UbiCollab main concepts .................................................................................................. 5
  5.1 Collaboration Instance ........................................................................................................ 6
  5.2 Spaces and Collaboration Spaces ....................................................................................... 7
  5.3 Services, Service Proxies and Service Domain ............................................................... 7
  5.4 Users and Virtual Identities ................................................................................................ 8
6 UbiCollab overall architecture.......................................................................................... 8
  6.1 Overall Functional Areas .................................................................................................... 9
  6.2 Application development framework ............................................................................... 13
  6.3 Component deployment and distribution ........................................................................ 15
7 UbiCollab ecosystem and value network ......................................................................... 18
8 UbiCollab implementation .............................................................................................. 20
9 Conclusions ..................................................................................................................... 20
10 Change log .................................................................................................................... 20

1 Cite as: “UbiCollab Architecture White Paper”, Babak A. Farshchian and Monica Divitini. IDI Technical Report 07/07. ISSN: 1503-416X.
1 What is UbiCollab?

UbiCollab is a platform for supporting collaboration on the internet. The advent of WWW and the new technologies related to Web 2.0 have made the internet a preferred platform for social interactions and collaborative activities across distances. Nowadays, nobody questions the ability of a web page being edited directly from a browser, an option that was impossible for only a few years ago. Moreover, social interactions are supported by such ubiquitous tools as email, instant messaging, software phones etc. Numerous online communities have emerged during the last couple of years, and in some cases made headlines because of their popularity.

Much of this advancement in technology is the result of visionary thinking in the field of CSCW and groupware technologies in the last couple of decades. This is also where UbiCollab, as a research project, brings its inspiration. UbiCollab is the next logical step in collaboration support in that it treats mobility and ubiquity as inherent properties of social interaction. Most groupware applications were designed to support desktop and web-based collaboration, where collaboration artifacts were digital, such as documents, and manipulated using windows-based GUIs on desktop computers. UbiCollab is an attempt to take the lessons learned from groupware research into a further step in order to naturally support collaboration in any situation the users are in. In doing this, UbiCollab draws upon research in the areas of user mobility, ubiquitous computing and pervasive computing.

UbiCollab provides a platform that captures the commonality of collaborative applications and provides generic mechanisms for applications to be built without extensive coding. UbiCollab tries to be domain-independent and provides only the basic functionality. UbiCollab is therefore following an open innovation approach where third party applications play an equally central role as the platform itself. Integration with physical environment where collaboration happens is a key aspect of UbiCollab.

2 UbiCollab Basic Concept: Human Grid

The abstract concept of a human grid constitutes the vision underlying UbiCollab (UC). A human grid is a collection of people and their artifacts/resources, connected together using UC platform technology. Interactions in a human grid are supported using the resources, artifacts services etc. imported into the grid by its participants. UC assists its users in building a human grid and supports communication in a human grid. Users in a human grid can be distributed geographically.
Human grid is adaptive and reconfigurable in that it will change its configuration in order to best fit context, services and artifacts that users have available in any given space. E.g. a user participating in a human grid at home or office will have different configurations of his/her collaboration spaces in these two locations, and UC will help automatically adapt to these different configurations as the user moves from one space to another.

3 UbiCollab Example Scenario

We will demonstrate UC functionality using one of its applications called UbiBuddy, a presence application that shows how presence information about people can be collected from physical spaces and distributed into physical spaces using physical artifacts. UbiBuddy, built as an application on top of UC platform, is representative of a very popular social computing application called a buddy list. Current examples of buddy lists are commercially available from AOL, MSN, Yahoo etc. and allow a user to keep an eye on the online status of his/her friends.

UbiBuddy demonstrates how the following shortcomings of available buddy list/presence applications can be overcome:

- Limited to desktop: Buddy lists are normally desktop-based presence applications (see Figure 3.A). In real life, presence information about users exists in their physical environment (e.g. turning on the light in your office tells your colleagues that you are at work), and users normally employ clues from the physical environment to derive presence information about their peers. Commercial buddy lists have few means for connecting to the physical environment of their users in order to collect or distribute
presence information. In this way, they have limited and unnatural ways of mediating presence information. Mobile versions of these buddy lists (Figure 3.B) also have limited access to physical environment, and are merely miniaturized versions of their desktop counterparts. UbiBuddy provides radical enhancements in that it allows users not only to collect presence information from available artifact in the users’ physical environment, but also to distribute presence information into these or other artifacts (Figure 3.C). UbiBuddy shows how UC platform functionality can assist easy development of this type of device-centric ubiquity.

- Flat social networks: All current buddy list applications support only flat social networks. Users are either online for all their peers or offline for them all. In reality, users’ availability for their peers, and their provided presence information toward these, depend on the context each user is in. For instance, when at home a user might not want to be available for work colleagues but available for family members (see Figure 4). UC ‘s Collaboration Instance concept (to be discussed later) is used in UbiBuddy in order to allow users define various contexts for presence. In this way, a user of UbiBuddy is actually adjusting his/her presence and availability relative to specific contexts and not relative to all his/her peers. In Figure 4 the user has three Collaboration Instances (“Work”, “Family” and “Football club”), and two Spaces (“Home” and “Office”). Depending on where the user is, he/she can selectively set his/her availability in different collaboration instances.

![Figure 3: UbiBuddy integrates with physical artifacts and devices surrounding its user.](image-url)
**4 UbiCollab design philosophy**

_TBA_

Show these principles using UbiBuddy as example.

- Collaboration is physical
- Collaboration might happen in physically distributed groups
- Collaboration rules are decided by collaborators
- Collaboration content is decided by collaborators
- Operational rules and infrastructure are decided by collaborators
- Collaboration is spontaneous
- Translucence is central
- Automation should be used with caution (starting point is the user’s actions)

**5 UbiCollab main concepts**

In this section we will describe UC’s main design concepts. Figure 5 below shows a UML class diagram illustrating the relationships among these concepts.
5.1 Collaboration Instance

The central concept of UC as seen from an application viewpoint is the Collaboration Instance (CI). CI is where users in a human grid (a group or a community) share information. Each human grid is implemented using one CI. Each CI has one owner and a number of participants. UC applications and tools (see Section 6.2) interact with CIs, visualize CI content in various ways, and manipulate this content. Each CI and its content can be used by multiple applications and tools (e.g. for providing customized views of the CI content for different users). In its most basic form, a CI contains the following information:

- Information about its participants.
- Information about Collaboration Spaces (to be discussed shortly) that its participants have registered.
- Information about Service Proxies (to be discussed shortly) that each participant has published in the CI, and how to access these (e.g. link, user name and password).
- A generic data repository for storing application-defined data about the CI (e.g. data necessary for implementing awareness and notification mechanism in UbiBuddy). This repository is used by the applications, and its content is defined by the applications.

UbiBuddy users in Figure 4 have defined three CIs (“Work”, “Family”, “Football club”). UbiBuddy allows its users to see presence information about participants in each of these CIs.
In addition, UbiBuddy allows its user to set presence information for himself/herself in each CI. All the information that is accessed or modified in a CI is immediately available to all the participants of that CI (through synchronization mechanisms in UC).

### 5.2 Spaces and Collaboration Spaces

Physical spaces and locations play a central role in UC and are used as resources for collaboration. A **Space** in UC represents a physical space with confines defined by a user. Each user has a set of Spaces representing different physical spaces that the user normally visits. Spaces are stored in UC using their geometric coordinates. A Space can be office, home, etc. When a Space is used as a resource for collaboration (e.g. shared with the participants in a meeting), it is called a **Collaboration Space (CS)** and is managed by the CI for that collaboration. Although each CS in a CI represents a Space defined by a single user (i.e. is linked to one of that user’s Spaces), the CS is adapted to a group context. For instance, “My office” might be a Space that user Tom created to denote his office. When Tom participates in a CI while sitting in his office, his office will be represented as a CS called “Tom’s office” in the CI and might have additional set of information/annotations useful for that CI’s participants. In the UbiBuddy example, as shown in Figure 4, the user has defined two UC Spaces, “Home” and “Office”. These spaces are used in the three CIs that the user owns or participates in. In each of these CIs, the Spaces defined by the users will be represented as CS called e.g. “Tom’s home” and “Tom’s office”.

Spaces are represented in UC using a mandatory set of parameters such as title and coordinates of the surrounding area, but also with a set of arbitrary information such as level of access, main purpose of the space, owner of the space etc. [REF]. This arbitrary information is defined and used by the applications. UC guarantees seamless access to the information no matter what the content is. A Space can be published in form of a CS in a CI. Having two distinct concepts for Space and Collaboration Space in UC (and different architectural management of these two concepts; as we will see in Chapter 6, Spaces are managed by a Space Manager, while CS is managed by CI Manager) is important because of a number of reasons. First, spaces often contain information that is in essence private. When using space as a resource in collaboration, only a subset of the information about the space is revealed to collaborating peers. Second, the interpretation of the same space is quite different as seen from an individual’s point of view (i.e. “My home”) and as seen from a group point of view. Having two separate concepts as we have decided to have in UC allows us not only guarantee user privacy at both conceptual and architectural levels, but also enable different interpretations (in form of annotations to a Space when represented as a CS) in different collaboration settings.

### 5.3 Services, Service Proxies and Service Domain

UC concepts of CI and CS allow a group of distributed users share an arbitrary set of information, and be aware of each other’s physical location. These two concepts support the idea of mobility in distributed online collaboration by allowing users be aware of each other’s location. A step further in supporting natural and ubiquitous collaboration is implemented by the UC concept of Services, Service Proxies (SPs) and Service Domain (SD). Our goal is to allow users deploy external resource in their collaboration with others. This will allow for a natural way of collaboration by for instance using dedicated devices and services in a meeting (such as projectors, whiteboards). In UbiBuddy, as shown in Figure 3, the user is able to use artifacts to signal presence information. For instance, a table lamp is used to signal the availability of a contact in UbiBuddy (by being turned on and off). A nabaztag rabbit is used for the same purpose, while a digital camera is used to take photos and share it with the other users. A GPS-enabled clock is in addition used to provide the user’s current GPS coordinates.
The notion of a **Service** is used in UC to denote such external resources brought into a UC environment in order to be used in collaboration. The mechanism used to connect to these Services (which might be devices, web services etc.) is through a dedicated **Service Proxy (SP)**. SPs are discovered using UC’s **service discovery** mechanisms (described later) by e.g. reading an RFID tag attached to the actual Service. This tag refers to a **Service Advertisement** used in order to dynamically locate, install and set up an SP at user’s wish. In order to facilitate the management of many SPs that a user potentially might have, each user is assigned a **Service Domain (SD)**. All the SPs installed by a user are registered and maintained by that user’s SD, which is also responsible for other tasks such as secure access to SPs and protection of user’s privacy. Each SP is in addition tagged using a **Space identifier**. This means that UC can support location-aware access to Services. For instance, if a user resides in a Space called “My home”, only Services labeled with “My home” might be available to that user’s applications by default.

Since the main purpose of UC is to support ubiquitous collaboration, we need to allow users share their Services with other users. This is done through a process we call **service publishing**. A user can choose to publish one of his/her Services in a CI. For instance, in the UbiBuddy example shown in Figure 3 the table lamp can be shared in a CI so that remote friends can use it to signal their availability to this user. Once a Service is published in a CI through its SP, the CI has a link to the SP and an appropriate pair of user name/password (assigned by the SD containing that SP) to access that SP.

### 5.4 Users and Virtual Identities

TBA

### 6 UbiCollab overall architecture

UbiCollab architecture follows the Service-Oriented Architecture (SOA) approach. UC is implemented as a collection of independent components in form of dynamically deployable services that can be deployed and used independently on a mobile device (UC deployment is described in Section 6.3). Each UC component is being developed to cover a very specific area of responsibility in UC. Components can be mixed and used together in different configurations (compositions) decided by the application using them. Only those components that are needed by a specific user (and his/her applications) will be deployed on his/her mobile device.

Table 1 below shows the current list of main UbiCollab platform components and their overall responsibility. These components are designed to operate “per user”, i.e. each user in UC will have its own instance of each of these components deployed on his/her mobile device. In addition to these per user components, a number of shared components exist not listed here but will be discussed briefly in Section 6.3.3. These shared components include e.g. a **User Management service**. The rest of this chapter will provide a more detailed description of some of UC’s functional areas (Section 6.1). We will then discuss how application development happens on top of UC (Section 6.2). In Section 6.3 we discuss some deployment configurations for UC.

---

2 There are a number of security and privacy issues related to allowing remote access to services. We are aware of these issues, but current version of UC does not support any access control once a Service is made available to others. Security and privacy constitutes an important part of our future research.
<table>
<thead>
<tr>
<th>Component name</th>
<th>Overall responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Discovery Manager</td>
<td>Is responsible for finding Services in the user’s surrounding. It gets a service query and returns a Service URI referring to a Service Advertisement. It relies on Service Discovery Plug-ins to find the URI through native discovery protocols (e.g. through RFID, Bluetooth, SLP, UDDI).</td>
</tr>
<tr>
<td>Service Domain Manager</td>
<td>Is responsible for installing service proxies, managing data related to service proxies, and answering queries about availability of service proxies.</td>
</tr>
<tr>
<td>Collaboration Instance Manager</td>
<td>Is responsible for managing a user’s CIs and the data shared in each CI. CI Manager is also responsible for enabling the sharing of CI object among CI members distributed across a network.</td>
</tr>
<tr>
<td>Space Manager</td>
<td>Is responsible for creating Spaces, managing existing Spaces, querying Spaces based on point coordinates and locate points in Spaces.</td>
</tr>
<tr>
<td>Service Ontology Manager</td>
<td>Is responsible for maintaining ontology data, and providing search and query mechanisms for UbiCollab service ontology. Currently only in design phase.</td>
</tr>
<tr>
<td>Service Composer</td>
<td>Is responsible for maintaining/validating/managing service compositions, for executing service compositions, and for sharing of service compositions.</td>
</tr>
<tr>
<td>Identity Manager</td>
<td>Is responsible for managing (creating, defining, deleting) virtual identities for a user. Currently only in design phase.</td>
</tr>
</tbody>
</table>

### 6.1 Overall Functional Areas

This section provides an overview of the three main functional areas of UC: service management, space management and CI management. Shared services such as User Management and Backup Management are discussed in Section 6.3.3. We will limit our description of the components to those that are fully designed and currently have a running version.

#### 6.1.1 Service Management

UbiCollab relies on external Services (e.g. devices, web services, artifacts) for supporting collaboration in online groups and communities. For this reason a large part of UC functionality is dedicated to **service management**. Service management in UC is related to three core areas: service discovery, service installation and service publishing.

#### 6.1.1.1 Service Discovery

Service discovery in UC denotes the process of gaining access to a Service’s **Service Advertisement** (SA) which is located at the **Service URI**. UC does not directly talk to Services, but uses Service Proxies (SPs) for this purpose. Any Service to be used by UC, such as a nabazag rabbit, is assumed to be tagged with a Service URI that points to that Service’s SA. The tag itself can be an RFID tag, a bar code, or a pointer stored in a central service registry.

Figure 6 shows the overall design of UC Service Discovery Manager and the process of discovering services [REF]. The main Service Discovery Manager module in this figure is responsible for registering queries from applications (step 1). In addition, it provides an API
for Discovery Plug-ins to read these queries (step 2) and add their discovered Service URIs (step 4) to be fetched by the applications (step 5). In order to allow multiple native methods of getting access to the Service URI (step 3 in the figure), UC Service Discovery Manager uses **Discovery Plug-ins**. This is shown in Figure 6 in form of RFID and UDDI Plug-ins. Current implementation of UC includes an RFID Service Plug-in that allows a user to read an RFID tag attached to a Service, and a UDDI-like service registry Plug-in that can search in a central service registry for queried service types.

![UbiCollab Service Discovery Process Diagram](image)

**Figure 6: Service discovery process in UbiCollab**

The retrieved Service URI points to the Service Advertisement (SA) that might reside on any web page. SA is an XML document that provides necessary information for talking to the Service (not shown in Figure 6, see next section). This information includes:

- Type of the service according to the service ontology.
- General Yellow Pages information about the service, what it does, who provides it etc.
- Pricing and authentication information.
- A link to a downloadable, signed SP that can be used to access that service from UC.

Although UC currently does not make extensive use of service ontology technology, we are in the process of enhancing UC, in particular its service management infrastructure, to be based on a formal ontology. This will allow more powerful queries and more accurate searches for services.

### 6.1.1.2 Service Installation

Once the Service Discovery Manager gets hold of the Service URI, this URI is passed to the application asking for the service. From this step forward UC’s Service Domain Manager (SDM) takes on the responsibility of downloading the SP and correctly **installing the service** in the user’s Service Domain. The process of service installation is illustrated in Figure 7. The application asking for a service installation provides the URI (obtained in step 5 of Figure 6) and a Space identifier (Space identifier can optionally be obtained from user’s Space Manager, step 0 in the figure. See Section 6.1.2). Service Domain Manager then accesses the SA (step 2 in Figure 7), analyzes the SA and downloads the corresponding SP (step 3). After the SP is installed (step 4) a link to the SP is returned to the application (step 5).
Ideally, the service installation process should be very quick. In its current implementation, UC does not do much more than downloading the SP and installing it on user’s mobile device. However, the installation of SP (step 4 above) is envisaged to include one or several of the following steps according to the type of the Service and the requirements of the user:

- Reading and accepting the terms of Service usage, and approving the digital signature attached to the SP.
- Deciding on the level of privacy, i.e. what identity to represent to the SP and how much personal data to enclose.
- Deciding service usage duration and providing mechanisms for paying for service usage (e.g. credit card number, account number). The user (or SP provider) might decide to delete the SP after the usage period is over, or keep the SP for next usage.
- Setting service-specific parameters and preferences.
- Assigning the service to a specific UC Space.

Although not all these steps will be necessary in all situations, the Service Proxy definition for UC allows third party service providers to implement any of these steps in their SP [REF]. Depending on the time it takes to install a service, UC can support many different scenarios ranging from:

- Nomadic: The user uses UC in specific locations, such as home and office only. In this case most services are discovered and installed in the first setup period, and new services rarely show up.
- Fully mobile: The user is constantly in new locations, confronting new services all the time. In these cases it will be an advantage to make the installation process as quick as possible.

Once a Service is installed, Service Domain Manager (SDM) API can be used to access the Service’s proxy. SDM can be used to view, delete, modify the description of existing SPs, and to assign and re-assign SPs to UC Spaces (e.g. in cases where Services are moved physically). SDM is also designed to protect the user’s privacy by assigning different identities to each SP usage. For instance, the UbiBuddy user can define a specific identity to represent the camera in Figure 3 in a specific CI. In this way the user can control how his/her services are
represented to different collaborators. UC currently does not provide access control at SP level, but this is part of our future research.

### 6.1.1.3 Service Publishing

**Publishing a service** in UC denotes the process of making the Service available in a CI in order to share it with other participants of the CI. Once a Service is published in the CI, not only the owner’s local applications, but also the applications of the other remote participants of the CI can access the Service through its SP. Publishing a service involves registering information about the Service (e.g. URL to its local SP, virtual identity information, password for accessing the service) in the CI it is published in.

### 6.1.2 Space Management

UC Spaces are representations of bounded physical locations that users normally reside in. UC Space Manager component allows users to define new Spaces based on their current locations (detected automatically using a location sensor or manually entered coordinates). Space Manager in UC can be used to locate users or services relative to already defined Spaces. For instance, an application can provide a set of coordinates (through e.g. a GPS sensor) and get back the relevant Space based on the set of Spaces that user has defined. Space Manager is currently used in these situations:

- Creating new spaces: Space Manager allows users to tag their current location as a new Space, add descriptive data for the Space, and register it for future use.
- Locating users: Space Manager accepts GPS coordinates for a user (acquired e.g. through a GPS sensor) and returns one or more matching Spaces. E.g. in UbiBuddy, each user is equipped with a GPS or other type of location sensor, which is discovered and installed in each user’s Service Domain as any Service. Applications can use this location sensor to get each user’s coordinates, and ask the user’s Space Manager for the correct Space for that user.
- Tagging SPs: Space Manager can be used to tag newly installed Service Proxies with a Space tag. Upon installation of new Services, users can choose to tag the SP with the correct Space identifier (through locating the user, and assuming that the service being installed is in the vicinity of the user). This is useful for embedded service such as a projector in a meeting room. This identifier can later be used by the applications to access physically collocated Services.

UC Space Manager already defines two default Spaces for its user: an Internet Space for online services (such as a file sharing service) and a Body Space for wearable Services (such as MP3 players carried around by the users).

### 6.1.3 Collaboration Instance Management

Collaboration Instances constitute a shared context (in form of shared information) for their users. The main job of the UC Collaboration Instance Manager (CI Manager) is to make sure that the information in a CI is always up-to-date for all users, and that only members of a CI can access this information. CI Manager also provides the main API towards the applications (see next section).

Figure 8 shows how CI Manager works. The rectangles denote CI Managers, and each user has his/her own instance running on his/her mobile device. For each CI that the user is an owner or member of there is a CI object (the round rectangles in the figure) maintained by that user’s CI Manager. Any changes made to a CI object by its owner are propagated to the CI Managers of all the members of that CI. The black arrows in the figure show the
synchronization paths. For instance Jan, Tom and Robert in the figure are members of the CI called “Football club”. Any changes done by any of them will be available for all the others and their applications. The same is true for Tom, Dennis and Robert regarding the CI “Work” and for Jan and Lisa regarding “Family”. CI Managers guarantee that no other user than CI members can see the data in the CI.

![Figure 8: CI Managers and communication among them. Black arrows denote data synchronization.](image)

CI Manager is a central part of UC. It is the main part of the architecture seen by the applications (see next section). A UC application will at least use the CI Manager. The other UC components such as Space Manager, Service Discovery Manager etc. are optional. CI Manager provides both a web service-based API and a call back function for applications to be notified of changes in the CI data. Applications can write arbitrary data to CI and be notified of changes to arbitrary data. CI Manager also includes a set of standard basic information (such as participant information, Collaborative Space information and Service information) in order to differentiate a UC application from any other application (although applications are not forced to use this basic information and can use CI Manager only for application-defined data exchange).

### 6.2 Application development framework

Although UC does not implement any specific applications (except a set of GUIs for interacting with UC components), the platform provides a set of facilities for applications developers:

- **Component APIs**: Each UC component has its own self-contained API. Applications can choose to use one or more of these APIs depending on their needs. The main API towards applications is provided by CI Manager. Additional APIs are available for Service Domain Manager, Space Manager and Service Discovery Manager.
- Application development framework: A set of guidelines and libraries are being developed to allow easy development of applications and tools (see below for a description of this framework).
- End-user service composition: Under development is also a set of graphical tools that allow end-users to compose UC components and services in different configurations in order to make customized applications/compositions [REF].

Figure 9 shows the main concepts in UC application development framework (second bullet above). The goal of this framework is to provide a set of basic building blocks that application developers can extend in order to implement their applications’ logic using UC functionality. These basic building blocks are not part of the UC platform, and are shown in gray in Figure 9. The overall approach is based on the MVC (Model-View-Controller) architecture with the CI as the model. Applications are views on the information available in a CI. An application can for instance be a visualization of a CI in form of a GUI or a web page. Tools are used to modify information in a CI, and to deploy the functionality provided by Service Proxies. For instance, a tool can query a user’s location from a GPS service, and update this location information in a CI so that all other participants can see the updated information.

A special type of application acts as a Portal, providing an overview of user’s applications. Portal is designed to be part of UC bootstrapping process and as the entry point to UC environment (a sort of mobile desktop). Portal provides the user with an overview of all CIs the user participates in or owns, all the applications that the user uses for viewing these CIs. Portal also provides access to a Toolbox that contains all configurations of all the tools the user uses with his/her applications. This Toolbox is responsible to make sure that the tools receive information and update information even if the user is currently offline in a CI.

UbiBuddy, discussed in Section 3, is implemented based on this framework following these steps:
The basic Portal is extended to display a window showing an overview of all the user’s CIs, together with their name, and a number showing how many users are online in each CI (Figure XX).

The basic Application is extended to show a window visualizing the contents of a CI. Double-clicking on a CI from the Portal view brings up the CI view, displaying all the online and offline users together with their locations, and their published services (Figure XX).

In addition, UC implementation includes a set of generic GUI-based tools for interacting with CI Manager (e.g. creating or deleting CIs), interacting with Space Manager (managing Spaces), and interacting with service management (discovery and installing services). For more information on these tools please refer to [REF] and [REF].

6.3 Component deployment and distribution
This section describes the physical deployment of UC components onto real network configurations. We will see where the different components of UC run and how they communicate with each other.

6.3.1 UbiNode
Each user in UC is represented by a mobile device called a UbiNode (See Figure 10). Currently users cannot be identified using more than one UbiNode. UbiNode is a network-enabled device that acts as a personal server, running a subset of the main UC components shown in Table 1 and some of the user’s applications. This means that each user has his/her own instance of a Service Discovery Manager, Service Domain Manager, Space Manager, CI Manager etc. running locally on his/her UbiNode. Not all the UC services need to be run by all the users. For instance, a user without a Service Discovery Manager can still install services manually in his/her service domain (by typing the URI). Or a user without a service domain manager can still use Space Manager and CI Manager together\(^3\). Complete independence among UC components allows us to outsource all composition tasks to the applications and guarantees a high level of modularity in the architecture of a UbiNode, in accordance with the SOA approach.

Figure 10 shows a typical deployment configuration on a UbiNode. In this particular configuration there are four UC components that reside in a container\(^4\). Each of them exposes a web service based API that is available to applications residing on the same UbiNode\(^5\). In addition, there are three Service Proxies installed on the node in the figure (denoted as white pentagons in the figure). Note that all Service Proxies, no matter which Space they belong to, are installed on the same UbiNode. Our future research on the Bootstrapping Service (see Section 6.3.3) will hopefully allow a more dynamic deployment of Service Proxies depending on the capabilities of the device and user’s location.

---

\(^3\) In the case of UbiBuddy this would mean that UbiBuddy supports multiple CIs and allows indicators for user location, but does not allow access to external devices.

\(^4\) Current implementation of UbiCollab is using OSGi as a container on user’s mobile device. See [http://www.osgi.org](http://www.osgi.org) for more on OSGi.

\(^5\) Note that applications don’t need to run on the device, but in most cases all or parts of a UC application will reside on the UbiNode in order to guarantee responsiveness. An application on a UbiNode can also be a proxy for a larger application in the back-end (such as a corporate application).
6.3.2 ServiceNode

A ServiceNode is where a service resides. Services are external to UC and are not directly accessed by UC but through their Service Proxies. Service Proxies are installed by Service Domain Manager and reside on user’s UbiNode (see Figure 10). A ServiceNode can be of the following types:

- A web service: A ServiceNode can be a web server providing access to a web service, such as a file server, a photo sharing service etc.
- A device: A ServiceNode can be a device a network interface, such as an X10-enabled lamp or a WiFi-enabled projector.
- A physical object: A ServiceNode can be a physical object such as a museum object. In this case the object is typically tagged using e.g. RFID tag, and the tag represents the Service Proxy for that object (in most cases this Service Proxy will be a simple web page).

6.3.3 UbiNetwork and UbiHome

In UC, all users sharing the same user management infrastructure constitute what we call a UbiNetwork. Figure 11 shows a sample UbiNetwork with four users represented by four UbiNodes in four different spaces (e.g. rooms). This UbiNetwork also has a shared UbiHome to be discussed shortly.
A UbiNetwork is mainly identified through a common service called **User Manager**, a trusted service responsible for managing (adding, deleting, verifying etc.) users in the UbiNetwork. In the UbiNetwork in Figure 11 the User Manager service is hosted by the UbiHome server. Once a user is registered in a UbiNetwork, User Manager for that UbiNetwork can be used to verify the authenticity of all the identities created by that user. A UbiNetwork can also exist without a centralized User Manager service but will lack the centralized trust support and has to rely on distributed trust (e.g. a reputation service). Additionally, two or more UbiNetworks can be **federated** through communication among their respective User Manager services. This latter scenario is useful when users are roaming in foreign networks enclosed by e.g. firewalls.

In addition to User Manager, a UbiNetwork can also have other public or shared services for the benefit of all the users of the network. Some of these services currently being designed are:

- **Rendezvous Service**: UbiNodes in a UbiNetwork can register their addresses with this service in order to allow for a centralized UbiNode register. This is useful when peer-to-peer discovery of UbiNodes is not possible.
- **Backup Service**: UbiNodes’ local data can be synchronized with Backup Service in order to safeguard users from mobile device damage and loss of data.
- **Bootstrap Service**: Similar to Backup Service, Bootstrap Service can be used to bootstrap a new UbiNode and set it up with user data and necessary components.
- **UbiNode Surrogate Service**: In cases when user is not carrying a UbiNode, or the UbiNode is turned off, this service can provide a virtual UbiNode that will play the role of the real UbiNode.
Although it is not mandatory, these shared services can run on a special type of node called a **UbiHome**. In its simplest form, each UbiNetwork will have all its shared services running on one UbiHome, as shown in Figure 11, which is typically operated by a dedicated UbiNetwork user. These shared services and the operation of a UbiHome provide an attractive business role for e.g. dedicated UbiNetwork operators.

### 6.3.4 Communication in a UbiNetwork

As shown in Figure 11, different types of communication are present in a UbiNetwork:

- **Inter-UbiNode communication**: UbiNodes communicate with each other mainly because of two reasons: 1) Communication among CI Managers residing on each UbiNode. A CI is a shared object, and changes done to its data is propagated to other UbiNodes sharing that CI. 2) Peer-to-peer communication among applications and tools. Communication tools residing on UbiNodes (e.g. instant messaging, video conferencing) communicate directly with other UbiNodes and/or exchange their data using CI Manager.

- Communication between a UbiNode and Services in its surroundings. This is mainly the communication that happens between a Service Proxy and its corresponding Service. In Figure 11 each UbiNode talks to the nabaztag rabbit in the same space.

- **Communication to shared services**: No matter whether there is a dedicated UbiHome in the network or whether one of the UbiNodes provides a shared service, a special type of communication happens between a UbiNode and the shared services. For instance, a Space Manager on a UbiNode might talk to a Backup Service to create a backup of its Space database. In Figure 11, where there is a UbiHome, this type of communication is illustrated between one of the nodes and the UbiHome server.

Comparing these types of communication shown in Figure 11 with the concept of human grid shown in Figure 2 illustrates clearly how UC sets out to realize the idea of human grid. The nodes in a human grid denote Spaces and Collaboration Spaces. Users reside in these spaces and bring in their resources into the human grid through UC service management infrastructure. The collaboration instance (in the middle of Figure 2) is implemented using CI Manager, while adaptability of the human grid to changing context of its users is enabled partly by Space Manager and partly by the applications. Shared services make sure that the infrastructure can be operated safely and smoothly.

### 7 UbiCollab ecosystem and value network

UbiCollab is built on an open business model and ecosystem approach where the goal is that the community of users will build the different parts of a UbiNetwork as needed by the community. UC is seen as a support and integration layer for what is already there, i.e. services and applications. UC is designed to provide the minimum necessary functionality in a robust and scalable way, without interfering with any application logic. It is our belief that a lot of future innovation will come from applications and services, and UC should not be designed in a way to hinder this innovation.

Figure 12 shows a birds-eye view of UbiCollab platform concepts (to the left) and the different roles involved in the development and operation of a UbiNetwork (to the right).
UbiNetwork Operator is the role directly involved with the UC platform. This role is in charge of hassle-free operation of the platform in a distributed and heterogeneous network infrastructure underlying the platform. UbiNetwork Operator (or operators) will be in charge of providing the shared services in a UbiNetwork, and for the correct operation of the UbiNodes. Although UC is designed with decentralized and user-centric operation as the first choice (i.e. end-users operating their UbiNetworks), we envisage that once such platforms are taken up by many users with various technical backgrounds an operator role will emerge naturally. UC is designed to optimize competition in operator services by 1) making the operator role optional (i.e. only if someone gains the trust of the users can that someone play the operator role), and 2) dividing operator duties into a number of distinct services e.g. user management, backup services, etc. (in order to allow each service be provided by the operator who is best in that area).

Besides the basic system functionality provided by UbiNetwork Operator(s), there are three important roles that are in charge of populating a UbiNetwork with innovative services, and adapting UC to the various needs of online communities:

- **Service Providers/Operators**: Network-enabled devices and services are operated by this role. This role will invest in services (e.g. devices) that are needed by the UbiNetwork, and will guarantee seamless availability of these services in needed locations. UC’s Service Proxy design makes it possible for a service provider to charge for the provided services, making this role a profitable and attractive one.

- **Service Proxy Developers/Operators**: A Service Proxy is the means by which UbiCollab accesses external Services provided by Service Providers, and by which applications talk to services (see Figure 12). Service Proxy Developers are in charge of developing high quality proxies for services. Service Proxy Developers might or might not be different than Service Providers/Operators. Service Proxies need to be trustable (i.e. signed) and can provided facilities to charge the user for service usage.

- **Application/Tool Developers/Operators**: In order to allow the specialization of UbiCollab to different collaboration domains, UbiCollab provides APIs to allow third parties develop applications and tools. Applications are in many computer science (and other) areas the part of the value chain that constitute the most of the innovation.
It is a central task of UC to provided high quality APIs that are easy-to-use and require minimum change to legacy code.

In addition to the various roles above, we have decided to make UC platform and APIs open source in order to allow community development of UC platform functionality. This means that the value network sketched above can easily be adjusted to the realities of UC users in cases where mismatches are identified (e.g. if more functionality is decided to be moved from applications into the platform).

8 UbiCollab implementation

UbiCollab project is a collaborative project between the Norwegian University of Science and Technology, Department of Computer and Information Science, and Telenor Research & Innovation. The goal of this project is the development and evaluation of UbiCollab platform technology. UbiCollab project is open source, and interacts with a number of European projects and with industrial partners internationally. For more information please visit www.ubicollab.org.

Currently UC is being developed in the Java programming language. The major part of the functionality related to dynamic deployment of services to devices and hosting of services on device is provided by OSGi [REF]. OSGi allows us to follow the SOA approach by being SOA-oriented itself. OSGi uses the concept of a bundle (a Java jar file) as the unit of deployment. Each UC platform component is being developed as an OSGi bundle. It is our goal that UC will support a number of other device-centric platforms in addition to OSGi (e.g. JavaME). We are basing our development on the open source implementation of OSGi called Knopflerfish. In addition we are using a number of other open source libraries for e.g. data replication. We intend to integrate open source identity management solutions (such as openID) as part of our future research. Currently alpha versions of the following UC components are available: Service Discovery Manager, Service Domain Manager, Space Manager and CI Manager. Some of the shared services are designed and implemented in a very simple form in order to allow us experiment with the main components. UbiBuddy is developed as a test application. UbiBuddy is developed in .Net in order to demonstrate the flexibility provided to application developers. UbiBuddy is also an open source project under the same license as UC.

9 Conclusions

TBA

10 Change log

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Changes</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>09.03.2007</td>
<td>1) added section 7.3 on applications and tools</td>
<td>Babak Farshchian</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2) Updated architecture chapter 8 based on latest agreements of subsystems.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3) Added a diagram for deployment in</td>
<td></td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Changes</td>
<td>Signature</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>0.4</td>
<td>22.05.2007</td>
<td>1) Worked on overall architecture and deployment.</td>
<td>Babak Farshchian</td>
</tr>
<tr>
<td>0.5</td>
<td>27.05.2007</td>
<td>Complete rewrite of the paper. Added extensive description of the architecture.</td>
<td>Babak Farshchian</td>
</tr>
</tbody>
</table>