NTNU
Innovation and Creativity

User-centered and collaborative service management in UbiCollab
Design and implementation

Kim-Steve Johansen

Master of Science in Computer Science

Submission date:	June 2007
Supervisor:	Babak Farshchian, IDI
Co-supervisor:	Monica Divitini, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

UbiCollab is a platform for supporting ubiquitous collaboration between people. UbiCollab is being developed in cooperation between the Norwegian University of Science and Technology, Department of Computer and Information Science, and Telenor Research & Innovation. The UbiCollab project is open source, and interacts with a number of European projects and with industrial partners internationally.

This task is related to service discovery and management, which is an important part of the UbiCollab's service oriented architecture. The candidate will propose and implement a solution for a user-centered and collaborative service management system for UbiCollab. The solution proposal will be founded on a previous study on this subject where a requirement specification and high level architecture were proposed.

Assignment given: 20. January 2007
Supervisor: Babak Farshchian, IDI

Abstract

This project has been carried out as a contribution to the UbiCollab project. The project aims to provide a platform for the support of ubiquitous collaboration. UbiCollab tries to support collaboration in the users' natural environment, and draws upon research in areas such as user mobility and ubiquitous computing to achieve this. The platform provides functionality such as location-awareness, integration with the physical environment and mobility support. UbiCollab is based on service oriented architecture (SOA), and integration of computerized services and service management are key aspects of the platform.

A previous pre-study has been performed by this author in the autumn of 2006 to compile a set of requirements and propose an architecture for a user-centered and collaborative service management system. This work builds on that study, and provides the design and implementation of a service management system for UbiCollab. The system aims to provide users with the tools to effortlessly discover, provide, and consume services. Users will also be able to take advantage of new services as they become available in dynamically changing environments.

Work done on the service management system consists of the design and implementation of several platform components and their application programming interfaces (APIs). In addition a set of applications to test the flexibility and functionality of the platform, as well as the completeness of the APIs, have been designed and implemented.

What sets the service management system in UbiCollab apart from similar systems is the focus on end-users and collaboration. User-friendliness is achieved by creating a pluggable service discovery system where the inherent complexity of service discovery protocols are hidden from the user. In addition discovery of services by pointing at the service of interest with an RFID device is supported. The use of pointing provides a natural way of communicating. Collaboration is supported by allowing users to share their services (publish) in defined groups, and consume services shared by other users.

Keywords: service discovery, service management, ubiquitous computing, user-centered, collaboration, UbiCollab

Preface

This thesis is submitted as the final work for the degree of Master of Science (Sivilingeniør) at the Norwegian University of Science and Technology. The report is based on work conducted from January through June 2007 on a project assignment given by the Department of Computer and Information Science (IDI). The work performed is a contribution to the UbiCollab platform. UbiCollab is an open source project with the aim of developing a technological platform for supporting mobile and ubiquitous collaboration.

In this report, work aimed at creating a user-centered and collaborative service management system for UbiCollab is documented. It extends work carried out on this subject in the autumn of 2006 by the author and Martin Børke. The report presents the design, implementation and evaluation of the selected solution.

The initial task description for this work is included in Appendix A. The formal task description has been updated during the semester, and the final task description is as follows:

> UbiCollab is a platform for supporting ubiquitous collaboration between people. UbiCollab is being developed in cooperation between the Norwegian University of Science and Technology, Department of Computer and Information Science, and Telenor Research & Innovation. The UbiCollab project is open source, and interacts with a number of European projects and with industrial partners internationally.
>
> This task is related to service discovery and management, which is an important part of the UbiCollab's service oriented architecture. The candidate will propose and implement a solution for a user-centered and collaborative service management system for UbiCollab. The solution proposal will be founded on a previous study on this subject where a requirement specification and high level architecture were proposed.

I thank my project supervisor, Professor Babak Amin Farshchian and co-adviser Professor Monica Divitini for providing input and ideas, as well as valuable feedback on the research and the writing of this report. My compliments also go to the students with whom I was fortunate to share office and collaborate with during my work.

Trondheim, June 17. 2007

Kim-Steve Johansen

Contents

List of Figures

List of Tables

Acronyms

AIDC	Automatic Identification and Data Capturing
API	Application Programming Interface
ASTRA	Awareness Services and Systems - Towards theory and ReAlization
GUI	Graphical User Interface
HTML	HyperText Markup Language
IrDA	Infrared Data Association
JVM	Java Virtual Machine
JXTA	JuXTApose (to set side-by-side)
LAN	Local Area Network
MMS	Multimedia Messaging Service
NTNU	Norwegian University of Science and Technology
OSGi	Open Service Gateway initiative
OWL	Web Ontology Language
PAN	Personal Area Network
RDF	Resource Description Framework
RFID	Radio Frequency IDentification
SLP	Service Location Protocol
SOA	Service Oriented Architecture
SRS	Software Requirements Specification
UC	UbiCollab or Ubiquitous Collaboration
UDDI	Universal Description, Discovery and Integration
UPnP	Universal Plug and Play
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WAN	Wide Area Network

WS	Web-service
WS-*	Web service specifications
WSDL	Web Service Definition Language
XML	Extensible Markup Language

Terminology

Architecture The conceptual design and fundamental operational structure of a computer system. It is a blueprint and functional description of requirements and design implementations for the various parts of a computing system.

End User The person who uses a product / system. The end-user may differ from the customer, who might buy the product, but doesn't necessarily use it.

Ontology A data model that represents a domain and is used to reason about the objects in that domain and the relations between them. Ontologies are used as a form of knowledge representation about the world or some part of it.

Resource A resource, or system resource, is a *physical or non-physical component* of limited availability. Devices connected to a computer system or communication network are physical resources. Internal system components are physical resources. Examples of non-physical system resources include files, network connections and services.

Service Contract A service contract is an agreement between the service provider and the service client, regarding how the service is allowed to be used.

Service Data Model The service data model is an abstract model of how services are related. The data model describes relations between the concepts, and each service may be mapped onto these concepts.

Service Oriented Architecture (SOA) A perspective of software architecture that defines the use of services to support the requirements of users. In a SOA environment, resources on a network are made available as independent services that can be accessed without knowledge of their underlying platform implementation. In SOA the interface definition hides the implementation of the language-specific service. SOA-compliant systems can be independent of development technologies and platforms.

Service Provider A service provider is an entity (person or organization) that provides services to other entities. Usually this refers to a business that provides web services to other businesses or individuals, but in UbiCollab a service provider is more often an end-user sharing his or her services with others.

Service Proxy Service proxies are OSGi bundles which run as services on the UbiCollab platform and provide an abstraction for different physical or logical devices used by applications which run on the platform.

Service Registry The service registry is the logical place where services are published to be discoverable.

Transparency In interaction, design transparency is used as a metaphor implying the level of understanding that an end user has in the use of a system. In human-computer interaction, the level of 'transparency' is a measurement of user friendliness and how much the user needs to worry about unnecessary or technical details (like installation, updating, etc). Transparency is to a large extent facilitated through the use of interfaces.

UbiCollab Platform The UbiCollab platform is the container onto which services are deployed, and it consists of core components such as the ontology manager, context manager and service discovery. In addition this is the container where service proxies are deployed

UbiHome A trusted centralised node in the UbiCollab Network (UbiNetwork). This node is an integrated part of the UbiNetwork and usually is the node which provides the centralised service discovery repository.

UbiNode A unit in the UbiCollab network that has installed the UbiCollab platform, and provides or uses services. UbiCollab users have their service proxies' installed on UbiNodes.

Chapter 1

Introduction

"Any intelligent fool can make things bigger, more complex, and more violent. It takes a touch of genius -and a lot of courage- to move in the opposite direction."
- E. F. Schumacker

People today live a very nomadic life where they daily roam between home, work, shops and leisure activities. The places they go to and the activities they attend are increasingly becoming more computerized. Both home and workplace environments are filled with computers and computerized services. It is common today to have embedded computers in items like telephones and cars, as well as household equipment like refrigerators and washing machines. All these computerized objects provide different kinds of services, some require a local presence while others are location independent. A common problem is however that while each service performs well on its own, interoperability is not easily achieved. The opportunities and challenges posed by such computerized surroundings are being addressed in the field of *ubiquitous computing*.

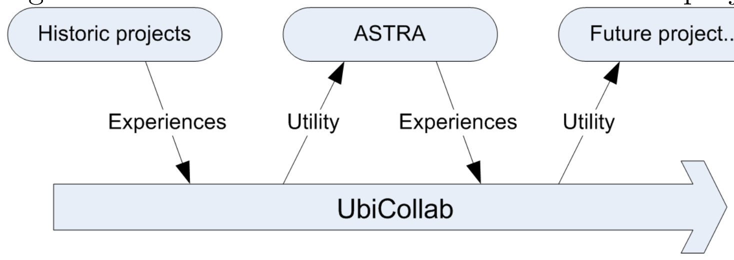
During the last few decades huge strides have been made in communication networks and especially mobile communication. Communication networks today reach into every corner of the world. The development and expansion of cellular networks, as well as other types of mobile networks, covers substantial parts of the technologically developed world. Where no other means of communication is available, satellite communication provides telephone and Internet access. The trend is that communication networks and methods are becoming faster, more ubiquitous and easier to use. This allows people to stay connected to each other and their desired online services regardless of location and time.

Since the 1990s devices like laptops with wireless LAN or WAN technology, Personal Digital Assistants (PDAs) with Bluetooth or IRDA interfaces, smart mobile phones and different kinds of wearable computers have been introduced. Such devices and technologies allow people to stay connected and access different services virtually regardless of where they are. The convergence of these increasingly pervasive computers and communication networks offer new opportunities and challenges for systems designers, and this is where UbiCollab comes into the picture.

UbiCollab is a research project which aspires to provide a platform where collaborative applications can be created without extensive coding [5]. The project has its roots in Computer Supported Cooperative Work (CSCW) and has been developed through iterative design and testing in various projects over the years. Today the intended application domain for UbiCollab has widened, and the focus has shifted towards Computer Supported Collaboration (CSC) in general. Some of the most important aspects of UbiCollab are to be domain independent, provide basic generic functionality, and provide integration with the physical environment. The aspect which sets UbiCollab apart from other CSC platforms is the focus on being general enough to be applied in all the different collaboration domains, while still providing enough functionality to be of practical use.

The UbiCollab platform is also being used by other research projects, such as Awareness Services and Systems - Towards theory and ReAlization (ASTRA) [1]. This is an ongoing project where among others NTNU is an participant. UbiCollab provides utility for projects like ASTRA and evolves with experiences received from them. How UbiCollab is used by research projects over time is visualized in Figure 1.1.

Figure 1.1: UbiCollab's relation to other research projects



During the time period when this work has been performed, UbiCollab has also moved on to become an Open Source project registered with Sourceforge[1]. The source code has been made available under the Apache License v2.0[2]. This move toward Open Source is expected to provide some benefits for the project, such as allowing former students to follow up on their work. In addition the usage of the platform should increase and this ought to generate more feedback which can be used to evolve the platform further.

The rest of this chapter is organized as follows:

In Section 1.1 the motivation for this work along with goals and contributions is described. This will give an overview of what is being accomplished by this work

[1]http://sourceforge.net/projects/ubicollab (Visited 04.06.2007)
[2]http://www.apache.org/licenses (Visted 04.06.2007)

and how this fits into the larger UbiCollab project.

Then, in Section 1.2 some important issues which need further research have been pointed out and research questions have been formulated.

Further, in Section 1.3 the research method which has been followed to achieve the goal of the project is described.

Finally, Section 1.4 describes how the rest of the report will be organized.

1.1 Motivation, goals and contributions

The original and the final task descriptions for this project are given in Appendix A. The final task description forms the basis for the project motivation, goals and contributions.

1.1.1 General motivation

The purpose of this work is to assist in the ongoing improvement of the UbiCollab platform by creating a solution for service discovery and management, suitable for use in a distributed collaborative environment. Service discovery and management is an important and necessary part of any Service Oriented Architecture (SOA), which is the architectural choice for UbiCollab. Dynamic discovery of services and devices is also a central enabler for pervasive and ubiquitous computing environments. Here, due to continuously changing context of the users, the system needs to find and adapt to new resource situations in the users physical surroundings. This work aims to create a solution to this problem which is user-centered and supports collaboration.

The rationale for creating a user-centered solution is to allow the end-users to take a more active part in the service discovery process and let them perform actions normally associated with service providers. In addition end-users will be given the tools to perform physical discovery of real world objects in their vicinity. When the user is more involved in this process, a better integration with the users' desires as well as the physical world is achieved. This approach is intended to increase the usability of the service discovery solution, which will also include a more traditional approach where services are registered in a searchable registry.

During the autumn of 2006 a project was carried out to create an architecture proposal for a user-centered and community oriented service discovery in a pervasive awareness system [10]. Various approaches to service discovery were analyzed, requirements were elicited and ideas for solutions was presented. The goal was to create a service discovery architecture suitable for the ASTRA project [1], which uses UbiCollab components in its architecture. The report from this project is accessible on the UbiCollab Web-site[1].

The work done in the autumn project has served as an important foundation for this project. In the reminder of this report, the resulting work from the autumn project will be referred to as the ***autumn report***. In addition to the work done in the autumn report there also exist a UbiCollab White Paper [5] which is important for this project. Also, much work on the platform has been done in earlier projects and has given important inputs and has served as an inspiration for this work. Most notably is the proof-of-concept porting of the UbiCollab platform from the old architecture onto a SOA architecture based on OSGi, as performed by Mosveen and Brustad in the spring of 2006 [16].

1.1.2 Project goals

The vision of UbiCollab is to provide a platform which is able to support a multitude of different collaboration domains. To be able to provide the necessary flexibility and extendibility, a Service Oriented Architecture (SOA) was chosen. Founding the system on SOA allows a very open system to be built where creative solutions can be created and tested, but at the same time poses several challenges. A prerequisite for such an approach is that service discovery, as an important part of SOA, can handle the challenges posed by the project objectives.

The goal of this work is therefore to design and implement a solution for service management which meets the requirements of the UbiCollab project. Much of the challenge is posed by the the focus on making the solution user-centered and flexible enough to support cooperation within a variety of application domains. This focus on a user-centered and flexible approach also constitutes the novelty of this work. The combination of these concepts has not received much attention in earlier research, as discussed in the autumn report [10].

To be able to accommodate a good solution for service discovery in UbiCollab, the problems associated with this task need to be elaborated and requirements extracted. A substantial part of this work has already been done in the autumn report, but there are still some aspects that need to elaborated further. Understanding the underlying requirements of UbiCollab is essential to tailor a service discovery module for the project, and failing to do so will have a major impact on the functionality of the platform. The problems associated with this task are further elaborated in Chapter 2.

1.1.3 Contributions

In this project a service discovery and management solution for the UbiCollab platform has been designed and implemented. This work presents the following contributions:

[1]http://mediawiki.idi.ntnu.no/wiki/ubicollab/index.php/UCSP_0052 (Visited 17.06.2007)

- State-of-the-art analysis
- Service Domain Manager design and implementation
- Implementation of a service discovery solution consisting of the following components:
  - Service Discovery Manager
  - Service Discovery Plugin for RFID
  - Service Discovery Plugin for Service Registry
- Service Registry
- Service Registry Management Tool
- Service Management Tool
- X10 service proxy (X10Proxy) and demonstrator GUI

The *state-of-the-art* analysis looks into relevant technologies which can be used to implement user-centered physical service discovery, and evaluates them for use in UbiCollab. This analysis is important to ensure the best available technology is selected and used in the solution.

Service Domain Manager is a Web service which is intended for deployment on UbiCollab nodes to provide management functionality for services residing on the node. This functionality includes start, stop, edit service attributes, uninstall and install services (by providing a URL for the service to be installed). In addition this service provides logging capabilities to ensure the person who owns services can see what is happening in his or her service domain.

The *Service Discovery Manager* is the central part of the service discovery solution. This is the component which service discovery clients connect to, in order to discover services. Also service discovery plugins, which are used to implement specific service discovery protocols, connect to this manager to provide search facilities in their domain.

The *Service Discovery Plugin for RFID* is a component which implements and provides RFID support for the Service Discovery Manager.

The *Service Discovery Plugin for Service Registry* is a component which provides the Service Discovery Manager with the ability to query the Service Registry.

Another contribution will be the *Service Registry.* This is a simple yet flexible centralized registry (repository) where services and resources can be advertised. It provides the functionality to add, edit and delete advertisements, as well as a query interface.

The *Service Registry Management Tool* is a simple tool which provide the Graphical User Interface (GUI) needed to manage a Service Registry. It supports querying of the registry, along with the functionality to add, edit and delete service advertisements.

The *Service Management Tool* is a tool which provides the functionality needed for an end-user to manage his or her services on a UbiCollab node (UbiNode).

This tool provides a graphical user interface which implements most of the functionality offered by Service Domain Manager and Service Discovery Manager. A user can for instance discover an interesting service through the use of Discovery Manager and then install this service by utilizing functionality offered by Service Domain Manager. This service can then be published to make it available for friends.

The *X10 service proxy* (X10Proxy) is a service which provides a Web service interface for controlling X10 devices[1]. A demonstrator GUI that can be used to demonstrate the functionality offered by the X10Proxy have also been created.

All these software components are designed and implemented based on the specification developed during the autumn project, and the findings in this work.

1.2 Research problems

Because this work is an engineering project, most of the focus will be on the design and development of the service management system. As explained earlier, much of the foundation for the design process will come from the autumn report, but there are also some additional problems that need to be researched further. An elaboration of the problems at hand has been carried out in Chapter 2. Based on this, one area which needed further analysis was pointed out.

In the computing environment where UbiCollab will be deployed, users own devices and services which they make available for other users in the collaboration instances where they are members, as described in the UbiCollab whitepaper [5]. Users will also perform "physical" service discovery on devices and artifacts they encounter, and suitable technologies for this have been selected for further analysis in Chapter 3. Physical service discovery refers to the ability of users to directly spot and physically select a service (e.g. by pointing at it) for being discovered by the system. This is enabled e.g. by visual tags or RFID tags. Based on the analysis, an evaluation of the candidate technologies were performed to determine which is best suited for the proposed solution.

The following research question has been formulated:

- Which technology is best suited for "physical" service discovery?

In addition to this question, the work performed in this project will also strengthen the answers provided to the research questions in the autumn report. The research questions from the autumn report were:

- What is implied by user-centered service discovery?
- What is implied by community oriented service discovery?

[1]X10 is a protocol which can send commands to compliant devices through the power grid

1.3 Research method

The research method for this project has been selected based on the need to get a thorough understanding of how service discovery should be implemented in UbiCollab, and to ensure the quality of the design and implementation. Based on the available relevant background material, the first part of the project was used to elaborate the problem domain and discover new areas that needed further examination. Next, a state-of-the-art analysis was performed to answer the question raised in the problem elaboration. The main finding of the state-of-the-art analysis was that service discovery technology is rarely designed to be used by average users.

From the findings in the problem elaboration and the technology state-of-the-art, a solution proposal has been created. In the solution proposal the design for the different components that will constitute the solution is outlined. Based on this, the implementation of the different components started as parallel activities. During the implementation the components were continuously evaluated until it was decided they fulfilled the specification set forth in the solution proposal.

The next and last phase was to assemble the different components and start testing and evaluation of the complete service management subsystem. The research method used in this work is illustrated in Figure 1.2.

1.4 Report outline

The rest of this report is organized into the following chapters:

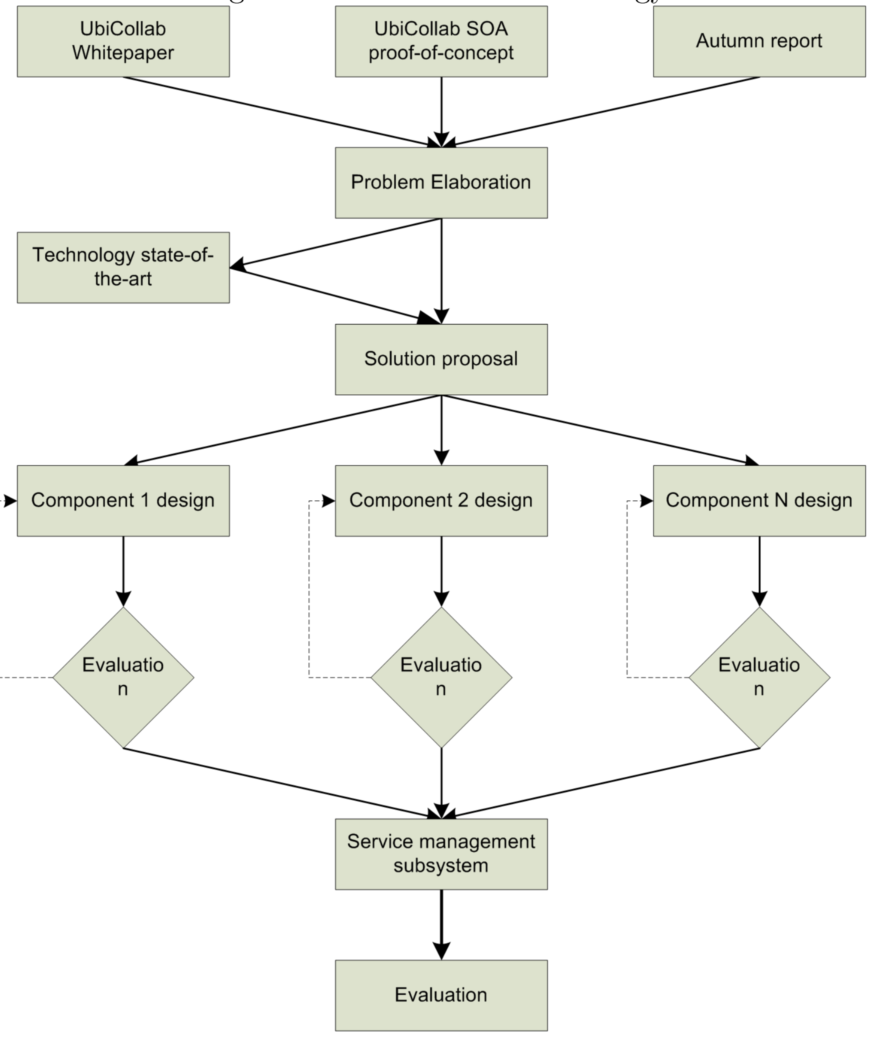
Chapter 2 gives an overview of important fields and concepts related to UbiCollab. The chapter then moves on to present a high-level UbiCollab user scenario that shows the utility of UbiCollab, before it further refines and analyzes it to find out which of the users' actions are related to service discovery functionality. Finally some important aspects related to the evaluation of the solution are pointed out.

In **Chapter 3** an analysis of three technologies that are deemed candidates for physical service discovery in UbiCollab is performed. Based on the analysis a comparison of the technologies is performed and one of the technologies is selected for the task.

Then, **Chapter 4** proposes a solution as to how the service management solution should be implemented, and which platform components are needed. This proposal includes the full service APIs for the platform components, and also describes how the functional requirements are being mapped to specific components. Components for testing and demonstrating the functionality of the solution are also described.

Next, **Chapter 5** presents how the proposed solution has been implemented. The implementation details for all components that have been developed are

Figure 1.2: Research methodology



presented, and certain key points about their inner functionality are described.

Further, **Chapter 6** describes how the solution has been evaluated, and explains what and how the solution have contributed to state-of-the-art. The evaluation is based on platform demonstrations where the created test applications and services are used, in addition to other work that uses the service management solution.

Finally, **Chapter 7** concludes the report by presenting the results and contributions. The work process is also shortly evaluated. Finally, some ideas and thoughts for future research projects which will provide value to the UbiCollab platform are presented.

Chapter 2

Problem elaboration

"*It is not enough to do your best: you must know what to do, and THEN do your best.*"
- W. Edwards Deming

The purpose of this chapter is to ensure that all the necessary research and analysis that will form the foundation for the solution proposal has been carried out. Much foundation work has already been carried out in the autumn report, but there are still some issues that must be covered before the proposal can be created.

This chapter gives an overview of the problem domain and discusses issues that need to be clarified in order to provide the desired service management solution. Some issues will be solved while other are pointed out for further analysis.

In Section 2.1 important concepts which constitute the foundation for UbiCollab are shortly described.

Then, in Section 2.2 a short summary of the history of UbiCollab is given.

Next, in Section 2.3 some scenarios have been created to demonstrate the utility of UbiCollab.

Further, in Section 2.4 the scenarios created are analyzed and their technical grounding are being described.

Moreover, in Section 2.5 the relation between ASTRA and UbiCollab is discussed, and some updates to the functional requirements elicited in the autumn report are described.

Finally, in Section 2.6 the problems involved in evaluating the solution are discussed.

2.1 UbiCollab foundation

In this section a brief description of some of the important fields and concepts which constitute the foundation for UbiCollab is given. These fields are Ubiquitous computing, Computer-supported collaboration and Mobile computing. A short introduction to Service Oriented Architecture (SOA) is also given, while a more in-depth description of these concepts are given in the autumn report [10].

Ubiquitous computing

Ubiquitous computing is a computing paradigm where technology and computational resources are integrated in everyday life and become a natural part of the environment [23] [16]. Preferably the computational devices should be accessible through "natural" interaction, unlike what is common today where humans to a large extent must adapt to the specifics of their desktop or laptop computers.

Ubiquitous computing is sometimes claimed to be the opposite of Virtual Reality (VR), because computers are integrated into the real world. In VR on the other hand, humans enter a fictional computer world. Other terms which are used to describe the same research field are *calm technology* and *pervasive computing*. Related research fields include Human-Computer Interaction and Psychology.

Computer-supported collaboration

Computer-supported collaboration (CSC) focuses on technology that enhances interaction in groups, organizations, communities and societies. CSC has its roots in the field Computer-supported collaborative work (CSCW) [4] but has expanded its focus to include all contexts in which technology is used to mediate communication, coordination, cooperation and even competition[1].

The applied side of the field has evolved from traditional groupware to online communities like MySpace[2] and FaceBook[3]. These are just two examples of the many online communities that have emerged during the last couple of years.

Mobile computing

The field of mobile computing focuses on the impacts mobility has on computational devices. Important issues that are introduced by mobility are for instance how communication can be established and maintained for mobile devices. Examples of other aspects addressed include power consumption, bandwidth and capacity. These and many other aspects are being dealt with in the field of mobile computing.

Users in the UbiCollab environment are highly mobile. They roam freely between networks and bring with them some personalized resources, while other resources

[1]http://www.acm.org/cscw2004/ (Visited 14.05.2007)
[2]http://www.myspace.com (Visited 14.05.2007)
[3]http://www.facebook.com/ (Visited 14.05.2007)

remain stationary. This implies that the address where UbiCollab services can be contacted will frequently change. This is an example of an issue that have been addressed in the field of mobile computing. Results from the field of mobile computing constitute an important foundation for UbiCollab.

SOA and Service Discovery in UbiCollab

SOA has been selected as the architectural approach for UbiCollab. This architecture utilizes loosely coupled services to provide the needed functionality [2] [10]. Some of the desirable aspects of SOA is that it promotes reuse of services, and is considered to allow developers to adapt quickly to changing conditions. Basic SOA is not tied to a specific technology, but in UbiCollab, Web-service technology has been chosen for the implementation. The platform (container) selected to deploy these services onto is OSGi.

Service Discovery is one of the cornerstones in SOA and provides the functionality needed to find services. In a traditional SOA environment a centralized registry is used to facilitate interoperations among applications. The problem with using this approach is that Service Discovery in UbiCollab aims to be user-centered, rather than application centered and centrally administered. Ordinary users with a high grade of mobility and living in a ubiquitous environment have other needs than applications in a business environment.

In addition UbiCollab aims to allow advanced end-users to share their services, which gives them the role of service providers. To be able to facilitate this role, adequate tools and functionality must be provided. A thorough analysis of the functionality needed for such a user-centered service discovery solution was performed in the autumn report, and will not be detailed further here. One important aspect of service discovery in a ubiquitous environment which has been given little attention earlier, is physical discovery of services. This aspect will be elaborated further later in the chapter.

2.2 The history of UbiCollab

UbiCollab as a project has been around for several years. In the spring of 2004 a prototype for a UbiCollab platform was developed by Schwarz as part of a master thesis [18]. This platform was later extended by work performed in other master thesises. The architecture for the platform at this time was a combination of loosely coupled UPnP components and client-server approach.

During the autumn of 2005 and spring of 2006 a new architecture was proposed for UbiCollab, and a proof-of-concept port of selected components was performed. This porting was to a SOA, using OSGi as container and Web-service technology for services. This work was done as part of a master thesis by Mosveen and Brustad [16].

Since then, a whitepaper describing concepts and the new architecture has been created by Farshchian and Divitini [5]. The project has also moved on to become open source under the Apache licence, and is currently available as a Sourceforge project[1].

At the time when this project started, no finished UbiCollab components existed but the underlying platform and architecture was established. In parallel with this project, two other master projects have also been working on other parts of the UbiCollab platform. These other projects are:

"Location awareness in UbiCollab", by Segelvik et al [19].
"Social tagging of services to support end user development in ubiquitous collaborative environments", by Laverton [15].

As we will see later, the work on service discovery has already been successfully used and evaluated by these projects.

2.3 A service discovery scenario

In this section, two short scenarios where UbiCollab is used as a natural part of everyday living have been created. The purpose of this is to demonstrate the utility of using UbiCollab, and how it can be used as a bridge between physical devices and computers in everyday situations. The scenario will also provide some examples as to how ubiquitous computing can enhance our daily lives.

2.3.1 Scenario 1

Jane sits at home in her living room watching TV, waiting for her son Jake to come home. Jake has been over at a friend's place, watching videos. It is getting late and Jane is starting to worry a bit as she so often does. Jake who is 16 years old, used to be embarrassed when he came home and his mother was waiting for him, thinking she was intruding on his privacy. To give Jake some more privacy, he got a room with his own entrance. The condition Jake had to agree on for this to happen, was that his mother could know when he came home at night. Jake who dreams of becoming a software engineer when he grows up, have configured UbiCollab so that his mother will get a message when he comes home.

After a while, Jane is looking at her watch and starts to get impatient. Then at last, a soft tone is played and Jake's picture appears on the coffee table in front of her. This signals that Jake has turned on the light in his hallway, and Jane can finally relax and focus on her movie again. Jane likes this new way of being informed when her son has come home, and especially the flexibility provided by the system. For instance, when she is working late she gets an SMS telling her that Jake has come home.

[1]http://sourceforge.net/projects/ubicollab/ (Visited 14.06.2007)

2.3.2 Scenario 2

Jake has been to school and is standing at the bus stop, waiting for the bus to arrive. When he looks around he notices a poster advertising a pop concert, hanging on the wall of the bus shed. He sees that the poster has a marker indicating that more information is available. Jake points his Service Popper[1] *towards it and almost immediately three services pop up on his PDA. The first service is a link to a web-page with information about the concert, the second is a service where tickets can be ordered and the last service provides the possibility to download sample songs that will be played at the concert.*

The last service seems most interesting for Jake, which uses it to download a couple of songs. He listens to the songs and finds out that he likes this music. However, Jake does not want to go to the concert alone so he decides to share the service with his friends to see if they also like the music. Jake then sends a message to his friends using UbiBuddy[2]*, asking them to check out the music.*

2.3.3 Scenario analysis

These two scenarios demonstrate several important aspects of Ubiquitous computing. In the first scenario the light switch in Jake's hallway is used to obtain information about his location. This information is then used to inform his mother that he has come home. This is an example of how physical devices can be integrated in everyday life, using UbiCollab technology. Jane's location is also used by the system to decide which method should be used to inform her. This is an example of how the system can adapt and use contextual information to produce a suitable response, and shows the flexibility of the system.

Scenario 2 shows how Jake can use a physical device discovery in order to interact with new services without having any foreknowledge about them. He did this by simply pointing his Service Popper towards the poster, indicating he wanted more information. At the same time Jake used his Service Popper to bridge the gap between the paper domain and the computer domain. Another solution which is in common use today would be to have a printed URL on the poster which Jake could enter into his PDA to produce a similar result. The key aspect that comes into play here is the level of user-friendliness. For instance, typing a simple URL like "http://www.google.com/" requires over 70 key presses with some phone models [20].

The different technologies used in the scenarios already exist. The smart table already exists[3] and a switch being turned on or off can be sensed using X10 devices. Gateways for sending SMS messages from computers have been around almost as long as SMS. Mechanisms like the one Jake used to discover the services are supported by most modern phones. Both a camera phone and the IrDA

[1]Physical service discovery device
[2]UbiCollab buddy list application with ability to show shared services
[3]http://www.microsoft.com/surface/ (Visited 05.06.2007)

interface which is ubiquitous on modern mobile phones can be used to implement similar functionality. The utility provided by UbiCollab is that it supports the integration of all these seemingly incompatible technologies, and provides a set of programming abstractions to application developers in order to hide the heterogeneity of the technology.

2.4 Technical grounding for scenarios

In this section we will look at what is needed with regard to service management to achieve a practical realization of the scenarios described in Section 2.3.

In Scenario 2 Jake discovers a set of services utilizing a physical discovery mechanism. The use of pointing is a natural way of communicating [20] [22]. Children in all cultures use pointing inherently to demonstrate interest in objects. Attaching tags to objects in our natural environment enables interaction by pointing. This is a very different approach for the discovery of resources as opposed to what networked service discovery protocols provide. As explained in the autumn report especially multicast based discovery works well in some scenarios but it suffers from several problems based on its reliance on network topology. For instance, it can be difficult to be sure that we access the printer in our hotel room and not the one in the next room. Moreover, places may contain physical artifacts such as paintings in a museum, whose web presence (in a museum guide application) are eligible for discovery but which may be hosted anywhere on the Internet.

For the above reasons, it is considered that while network discovery has its uses, the physical discovery mechanisms of direct and indirect URI sensing are set to prevail in many practical circumstances. This implies that some form of physical discovery should be supported by UbiCollab. However, there exists many different technologies which can be used to accomplish this, and they all have inherently different characteristics. To decide which technology should be selected for implementation in UbiCollab, a survey needs to be carried out. This survey is performed in Chapter 3.

Scenario 1 makes use of an already configured UbiCollab system. The part that is interesting here with regard to service management is how such a system can be installed and configured by the end user (since we are aiming at empowering the end user), before it is put to use. Work related to this has already been carried out in the autumn report, but the aspect of installing services was not considered part of the service discovery process. This resulted in that service installation was only part of the analysis, and was not included in the requirement specification. The relation between installing services and the service discovery process was acknowledged, but how services should be installed was not covered in great detail. Section 4.1.3 in the autumn report gives some suggestions as to how a user-friendly service installation should be performed, but does not go so far as to elicit requirements.

In order to ensure that an adequate solution is proposed, the functional requirements related to installation and management of services also need to be derived. The implication of this is that the relation between UbiCollab and ASTRA, for which the requirements in the autumn report were elicited, should be reviewed. This will help to ensure that all the necessary functional requirements are attended to. The analysis performed in this section should also be taken into account when the requirements are updated. The next section describes the relation between UbiCollab and ASTRA, and how the requirements are updated.

2.5 Relations between UbiCollab and ASTRA

The relation between UbiCollab and ASTRA is important for this work, as the autumn report was founded on the ASTRA scenarios. The reason for this was that ASTRA worked on scenarios during the autumn, and these scenarios were used as a source for the functional requirements. ASTRA architecture (as developed in parallel to UbiCollab during this spring) is very similar to UbiCollab's, but there are some differences in vocabulary and concepts between them. ASTRA seeks to support collaboration in communities, while UbiCollab seeks to support collaboration in general. ASTRA can in a sense be seen as a specific application of UbiCollab.

Another important aspect between the autumn report and this project, is that the autumn report focused on creating a solution for service discovery while this project aims to create a solution for service management. Here, service management includes both service discovery and management of services. In the autumn report, the functional requirements for management of services was not assumed to be part of the proposed architecture. An analysis of management was performed in Section 4.1.3 in the autumn report, but functional requirements were not elicited.

The reason for adding management issues to the mere discovery of services is that we see the need to enable users to easily manage the potentially large number of services they will discover and use in the course of interacting with a pervasive environment. For this reason, the functional requirements from the autumn project have been updated to comply with the current task assignment. This update includes some rephrasing of requirement wording because of differences in vocabulary and concepts between UbiCollab and ASTRA, and adding of requirements for service management. The updated functional requirements have been incorporated in the solution proposal and can be found in Appendix B.

2.6 Evaluation of implementation

A thorough evaluation is important in order to ensure that the implemented solution fulfills the specified requirements. The solution will be evaluated against the revised list of requirements presented in Appendix B. In addition to performing an evaluation against the requirement specification, the solution should also be evaluated against how well it is able to fulfill the end-users needs and expectations. One can never guarantee that the requirement specification is complete and that it has anticipated what the end-users really need. To cover this aspect some scenarios have been created and played out, and how well the solution perform are evaluated. By evaluating this way not only the solution is evaluated, but also the requirement specification.

A practical solution to how this evaluation can be accomplished must also be outlined. This is a common problem when designing and implementing platforms and platform components, such as the service discovery subsystem. There exists established and reputable techniques for designing and evaluating applications made for a specific purpose, whereas techniques for evaluating platforms and platform components are not so common or well defined [3]. This is a consequence of the difficulty introduced by anticipating the requirements from end-users and applications that will use the platform.

Evaluation of platform components are difficult because most of these are not visible to the end-user, and still the purpose of these components is to enhance the end-user experience. This implies that applications and tools that use these components must be created, and an evaluation can be performed based on how well the platform supports these. The software components created for this purpose will also be useful for platform demonstrations, and as such will provide value beyond evaluation purposes.

Another element that should be evaluated is how well the newly created components work with other platform components. Evaluation of this is a key issue to ensure full compatibility among the platform components. This will also help to assure that application developers get a uniform platform to work with. For this reason it is important to collaborate with the other groups working on UbiCollab, and create applications which span across all our work. Such applications will also aid to demonstrate the utility of UbiCollab.

A complete evaluation of this work, including the issues pointed out above, is provided in Chapter 6.

Chapter 3

Technology State-of-the-Art

"Research is what I'm doing when I don't know what I'm doing."
- Wernher von Braun

This chapter will describe technology which is of interest for implementing a physical service discovery solution for UbiCollab. Much work on service discovery in general has already been done in the autumn report, such as creating a taxonomy, describing related research and evaluation of common service discovery protocols. In addition, the built-in service discovery in OSGi was analyzed and deemed unsuitable for use in UbiCollab. This chapter should be seen as an extension to the state-of-art work performed in the autumn report.

The aspects of service discovery where end-users physically discover services or resources was mentioned in the autumn report, but no further research were performed. Because of the focus on integration with the physical environment and end-users in UbiCollab, this aspect needs to be investigated. The motivation for doing this is to get a better understanding of the existing relevant technologies, and discover which technology is better suited to solve the challenges at hand.

From the many candidate technologies which could potentially be used, three have been further analyzed. These are IrDA, Optically Readable codes (barcodes) and RFID. From these RFID has been selected because of its high degree of user-friendliness and versatility. In addition this technology is slowly becoming ubiquitous and is also considered very promising for service discovery application in many different areas.

In Section 3.1 three different technologies which can be used to perform physical service discovery are investigated and several key aspects are pointed out.

Then, in Section 3.2 a comparison and evaluation of the technologies investigated in Section 3.1 is performed.

3.1 Physical Service Discovery

In this chapter three technologies which are considered candidates for the physical service discovery solution in UbiCollab will be examined to find out which is better suited for the task. These candidates are IrDA, Optically readable codes and RFID. They have been selected from a group of technologies often referred to as Automatic Identification and Data Capture (AIDC), or Auto-ID technologies [7]. An Auto-ID technology is one that may be used to automate the identification of objects.

The common denominator for AIDC technologies is that they can be used for identification of objects by storing and reading a code, data or other information. Such technologies are used in application areas like tracking goods, security purposes, check-out points, identifying objects and many other areas. The selected technologies have all been used for physical service discovery purposes in related research, experiments or in the field.

AIDC technologies can be used to minimize user intervention in the identification of objects, thus allowing the user to accomplish his task with a minimum of effort. Some AIDC technologies in common use today include: optical readable codes (e.g. barcodes), magnetic stripes, IrDA, smart cards, RF remote controls, RFID systems, ultrasound systems and visual identification. Other wireless technologies like Bluetooth have also been considered, but the ones selected were deemed most promising. All three technologies selected for further examination have been used in research for similar purposes earlier.

3.1.1 IrDA

The Infrared Data Association (IrDA)[1] is the organization responsible for the development and maintenance of the communications protocol standards often referred to as the IrDA specification. IrDA was originally launched as a cable replacement technology in 1993, but has evolved with time. The specifications today define a layered model for short range exchange of data using infrared wireless communication (infrared light) [13]. The IrDA model is comprised of three mandatory layers (Physical, Link Access and Link Management) and a wealth of optional protocols for various application domains on top.

For many years the use of IrDA was not very widespread. IrDA first started to gain ground after IrDA-enabled PDAs introduced the possibility for end-users to easily swap applications and information. This is still the most common application area for IrDA. Today, virtually every PDA supports IrDA, as do many mobile phones, laptops, printers, and other products. This widespread use has lowered the cost for the components needed for implementation, to approximately $2 according to www.irda.org as of 17. April 2007.

The range between two IrDA communication devices is normally limited to 1

[1]http://www.irda.org (Visited 17/04/2007)

meter and requires a direct line of sight. When using low power devices this range is reduced to 0,2 meters. The specification also requires the communicating IrDA transceivers to communicate in a directional matter, with a cone that extends minimum 15 degrees off center. Currently, available speeds range from 2,4 kbit/s to 16 Mbit/s, while an Ultra Fast Infrared (UFIR) protocol offering speeds up to 100 Mbit/s is under development.

Probably the most notable research where IrDA was used for discovery of resources, was performed by HP Labs and was named *Cooltown*[1]. The purpose of this experiment was to bridge the physical world with the World Wide Web by giving real life objects a web presence [12]. This was done through the use of infrared markers on or in close proximity to physical objects. Such markers could contain URLs to web pages with more information about the physical object, or related services. Two different types of markers were used, “beacons” which actively emitted its information every 3 seconds and “tags” which had to be activated. Both kinds of markers could be read by PDAs and other IRDA-enabled devices.

Other related or similar experiments have also been performed. In one example regular end-users employed IrDA-enabled mobile phones to interact with physical objects [22]. Test cases included accessing web resources related to physical objects, as well as invoking phone calls from pointing at pictures. Most test subjects reacted positive to this way of interacting with a ubiquitous environment.

In both experiments IrDA was not directly related to the research goal, it was merely used as the enabling technology. Perhaps the single most important reason for selecting IrDA was the rich availability of IrDA-enabled mobile devices. The purpose was not to prove that IrDA was the best or most suitable technology for use in the experiment, but still it was proved suitable for the job. Despite of this, IrDA has still not moved out of the laboratory as a tool for general interaction with physical objects.

3.1.2 Optically readable codes

Optically readable codes are not a specific technology but rather a collection of related technologies often referred to as barcodes. The common denominator for these technologies is the use of machine-readable visible symbols on surfaces. Symbols are used to represent information and are created according to technology specific encoding schemes called *symbologies.* Such codes come in a variety of forms and shapes but all of them share the same basic principles of the original barcode where thick and thin lines were used to represent numbers.

Barcode as a technology was first patented in 1952, but it was not before 1970 that UPC[2] was selected as the standard and the wheels began to turn [21]. By the early 1980s, barcodes and scanners had become pervasive in supermarkets.

[1]http://www.hpl.hp.com/archive/cooltown/ (Visited 20/04/2007)

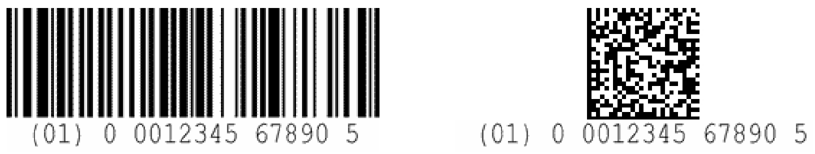
[2]Universal Product Code, introduced by International Business Machines (IBM)

Soon after barcodes became the standard Auto-ID system in other areas like the postal system and even on library books. Today barcodes are scanned about five billion times a day, and greatly simplifies information collection, processing and tracking.

Significant factors explaining the barcode success story are ease of use and automation. Barcodes greatly increses the speed of transactions and reduces likeliness of typing errors. In order to use barcodes, a infrastructure with a scanner (reader) and printer is needed. Prices for these ranges from about one hundred to several thousand dollars at Amazon.com electronics[1]. In general, barcodes are inexpensive and new labels can easily be printed on-the-fly.

Barcodes are commonly classified into three different types, which are linear (classical 1-dimensional barcodes), stacked (vertically stacked linear) and 2-dimensional barcodes (also known as 2D codes or matrix codes). Linear and stacked codes are easily read by LEDs or lasers in a sweep pattern, while stacked codes offer a higher data density than linear. The introduction of 2D barcodes offered a substantially higher capacity for encoding data, but also required reading by camera capture devices. This characteristic makes 2D codes suitable for reading by camera phones (which are becoming increasingly ubiquitous), and hence is suitable for use in a ubiquitous computing environment. Figure 3.1 visualizes the difference between 1D and 2D barcodes.

Figure 3.1: Information encoded using 1D and 2D barcodes



There exists several 2D barcodes like CyberCode[2] and Semacode[3] which are created for use with camera phones. Perhaps the most interesting one is the Quick Response Code (QR Code). The QR Code is a two dimensional barcode created in 1994 by the Japanese corporation Denso-Wave. It was released as a Japanese standard in 1999 and a corresponding ISO standard[4] was approved in 2000.

Employment of QR Codes has become incredibly popular in Japan and can be found everywhere, relieving the user of the tedious task of entering data into their mobile phone. A host of content types are defined in addition to URLs, like business card details, birthdays, email addresses etc. QR Code software is integrated in most new Japanese mobile phones and is said to have "become the door to the mobile Internet for the average mobile user"[5]. QR Codes have only recently been put to use in Europe and a free QR-Code reader which works on most newer European Java-enabled GSM phones can be downloaded from the

[1]http://www.amazon.com (Visited 29.04.2007)

[2]Visual tag system for augmented reality created by Sony.

[3]Code used to encode URLs for applications using cameraphones.

[4]International Standard ISO/IEC 18004

[5]http://harper.wirelessink.com/?cat=102 (Visited 28.04.2007)

Kaywa website[1].

One important reason for the popularity of the QR Code is the high speed reading/decoding (hence Quick Response Code). In addition it offers many interesting features which gives it a high degree of flexibility. Some of the characteristics of the QR Code are:

- **Support for omni-directional (360 degrees) decoding**. This mean you do not need to worry about the orientation of the code.
- **Space saving**. The 2-dimensional QR Code allows the same amount of data to be encoded in approximately one-tenth the space of a traditional bar code. When a smaller printout size is needed, a low capacity version named "Micro QR Code" is available.
- **Advanced error correction capability**. The Reed-Solomon[2]) algorithm allows decoding when up to 30% of the code to be damaged.
- **Flexible structure**. A single QR Code can be divided into a maximum of 16 smaller codes which can be structured, for instance, on a line to be printed in narrow areas.
- **Flexible data capacity**. The QR Code defines many symbol sizes, from Version 1 (21 x 21 modules) up to Version 40 (177 x 177 modules).
- **Large data capacity**, as shown in Table 3.1.

Character set	**Capacity**
Numeric only	7,089 characters
Alphanumeric	4,296 characters
Binary (8 bits)	2,953 bytes
Japanese (Kanji/Kana)	1,817 characters

Table 3.1: QR Code Data capacity

An example showing what a QR code looks like is displayed in Figure 3.2.

Figure 3.2: "http://www.ubicollab.org" encoded as QR Code

Limitations of the QR Code include distance and direction constraints, and also the requirement of a high quality camera. Another code which tries to alleviate these limitations is the "Visual Tag" [9]. Visual Tags promise a high level of

[1]http://reader.kaywa.com/getit (Visited 28.04.07)

[2]A mathematical error correction method originally developed to handle communication noise.

robustness with regard to illumination, distance, direction and quality of the camera used. This is achieved by using colored rectangles placed in a geometrically invariant arrangement, which puts very little constraint on the capturing. Unfortunately the algorithm needed for decoding Visual Tags is to computationally demanding to effectively run within a normal mobile phone.

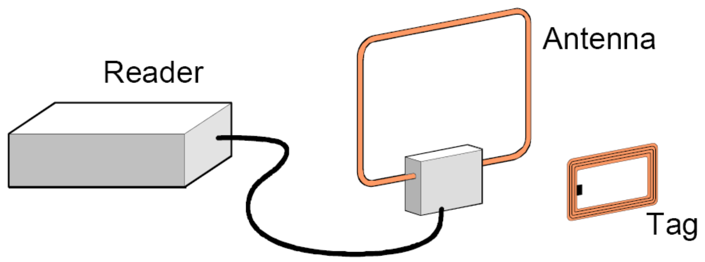
A similar technology is being developed by Microsoft[1] for commercial purposes. In the autumn of 2007 Microsoft is set to introduce the High Capacity Color Barcode (HCCB) technology which uses a multicolour symbol to code roughly double the amount of information per surface area, as current barcodes are capable of.

Another barcode project with a somewhat different technological solution is the "CybStickers" project[2]. CybStickers is a collaborative effort between SINTEF, Telenor R&I[3] and Netcom[4]. This project uses the camera on MMS compatible mobile phones to capture 2D barcodes placed on things/places, which are sent in a MMS to a provided number. The recipient resolves the 2D barcode and returns a corresponding message, linking things/places with information.

3.1.3 RFID

Radio-frequency identification (RFID) is a technology which relies on radio waves for remotely retrieving or writing data to RFID tags [6]. An RFID tag is an object that can be attached to or incorporated into almost any product, animal or person and is able to store data and be interrogated by an RFID reader. An RFID reader (often referred to as an interrogator) consists of a processing part and an antenna. Both tags and readers come in all kinds of different forms and shapes for different applications. The basic concept behind RFID is visualized in Figure 3.3.

Figure 3.3: The basic RFID concept



The history of RFID dates back to early in the 20th century and is tightly coupled to the development in radar technology, with one of the most important landmarks being a paper by Harry Stockman, "Communication by Means of Reflected Power" [14]. During the 1960s applications field trials started and development in

[1]http://research.microsoft.com/research/hccb/ (Visited 30.04.2007)
[2]http://www.sintef.no/content/page1___6785.aspx (Visited 14.06.2007)
[3]http://www.telenor.no/fou (Visited 14.06.2007)
[4]http://www.netcom.no (Visited 14.06.2007)

the field accelerated. However, it was not until the 1990s that standards emerged and RFID became widely deployed. Today much of the issues related to RFID revolves around global standardization, as the technology itself has become quite mature and boasts many impressive characteristics such as:

- The reading distance for passive tags can vary from millimeters to around 10 meters. There exists active tags that can be read at least a hundred meters away.
- Different technologies and frequencies can be selected for various application areas.
- Tags does not have to be in the line of sight of the reader.
- Non-contact read and write capabilities.
- High data storage capacity.
- Functionality in hostile environments (tags can be encapsulated in protective coating).
- Many commercial readers support "bulk scanning", which allows for 100+ tags to be read every second.

Many of these characteristics are desirable for businesses, and as such much of the current thrust in RFID development are funded by large enterprises and aimed towards use is in supply chain management. One example of such a thrust is the cooperation between Intel and Microsoft aiming to create a layered RFID infrastructure based on SOA and Web-Service standards (WS-*)[1].

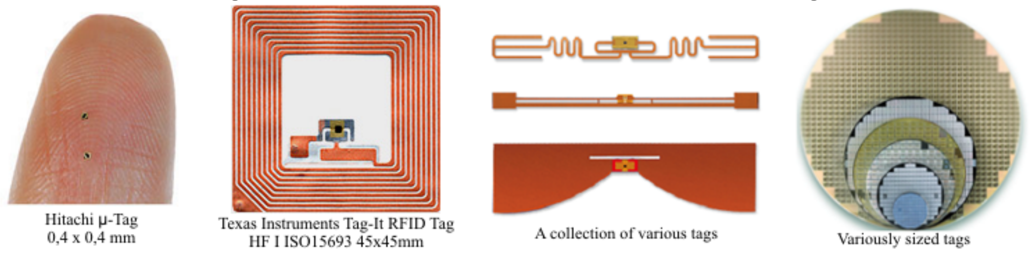
RFID Tags

Tags come in two fundamentally different versions known as active which is self-powered (usually a battery) and passive which relies on rectification of the emitted power from the reader's radio communication. Passive tags are most used and are commonly referred to as transponders. Active tags are less used, likely because they are significantly larger and much more expensive than passive ones. The smallest passive tag in common use today is produced by Hitachi[2] and measures 0,4x0,4 mm, while much smaller tags exist as prototypes. Tags come in a variety of forms and shapes as depicted in Figure 3.4.

As RFID technology matures, tags become increasingly cheaper and smaller. Prices of passive tags range from about 10-20 cents for simple read-only tags to 10-20 dollars for more advanced tags with a high data capacity. One of the things that hinders RFID from becoming even more ubiquitous is the complexity, prise, power consumption and size of the RFID readers. Another hindering for RFID has been the lack of internationally approved standards and interoperability. However, over the last few years this problem has decreased. A survey of RFID Standards is presented on the RFID Handbook Web site[3].

[1]http://www.microsoft.com/biztalk/technologies/rfid/default.mspx (Visited 31.05.2007)
[2]http://www.hitachi.co.jp/Prod/mu-chip/ (Visited 30.05.07)
[3]http://www.rfid-handbook.com/rfid/standardization.html (Visited 02.05.2007)

Figure 3.4: A collection of different RFID tags.



Applications of RFID

Even if the much focus is on businesses and inventory tracking, RFID technology is becoming increasingly ubiquitous and is starting to appear in other areas, like mobile phones and PDAs. Today, nearly everyone are using the technology on a daily basis without thinking about it [14] [6] [8]. Some of the application areas include :

- Identification badges and access control for equipment and personnel.
- Parking lot access and control. You must have the correct tag for a given parking space.
- RFID tags are commonly used in car keys as a anti theft device, preventing the car from starting without the correct RFID in near proximity.
- Implanted RFID tags are also used for animal/pet identification (typically dogs and cats), and livestock inventory control.
- RFID and can also act as a security device, taking the place of the more traditional electromagnetic security strip.
- Active RFID tags also have the potential to function as low-cost remote sensors that broadcast telemetry back to a base station.
- In an effort to make passports more secure, several countries have implemented RFID in passports.
- RFID is being used to tag humans in similar fashion as animals.
    - The Food and Drug Administration in USA has approved the use of RFID chips in humans.
    - Some high profile night clubs use an implantable chip to identify VIP customers. Customers use it to pay for drinks.
    - In Mexico the attorney general and 160 members of his staff carry RFID security pass under their skin[1].
- In Oslo, Norway, the upcoming public transport payment is to be entirely RFID-based[2]. The system is to be put into production somewhat delayed in the spring of 2007.

[1]http://www.msnbc.msn.com/id/5439055/

[2]http://www.rfidjournal.com/article/articleview/1975/1/1/ (Visited 02.05.2007)

- In Norway, all public toll roads are equipped with an RFID based payment system known as AutoPass[1].

New application areas for RFID systems continuously appear. One example is the *magicmirror*[2] which is a "smart mirror" equipped with an RFID reader behind it. The idea is to use item-level tagging in clothing shops and let the mirror display rich, user-centric information when a tagged piece of apparel is brought within range. The RFID tags on the garments can also be used for payment and as an anti theft device to add to the utility.

RFID technology is also starting to make it's entry into mobile phones. This has been a reality in Japan for some time, but now also European manufacturers like Nokia are starting to deliver mobile phones with integrated RFID reader/writer[3] support.

Privacy

Privacy issues have also emerged along with RFID systems becoming more ubiquitous. Illicit tracking of RFID tags can potentially be used to forge identity papers (e.g. passports), infer information about people and even real time identity theft. To counter such efforts, RFID protocols based on both symmetric and public key cryptography have been developed. Unfortunately, cryptographically-enabled tags come at a high cost and with increased power requirements. Another approach is a system utilizing *blocker tags* for "selective blocking". This has been proposed as a way of protecting consumers from unwanted scanning of RFID tags attached to items they may be carrying or wearing [11].

RFID in research

RFID have also been used as a physical discovery technology in various research work. One example is a computer supported ubiquitous learning environment for language learning named Tag Added learNinG Objects (TANGO). The system detects the objects around learner using RFID tags and provides the learner the educational information about the objects [17].

The future of RFID

The pace of developments in RFID continues to accelerate and the future looks very promising for this technology [14]. The achieve its full potential it also requires advancements in other areas as well such as:

- Development of applications software, e.g. on the UbiCollab platform.
- Careful development of privacy policies and consideration of other legal aspects.

[1]http://www.autopass.no/ (Visited 02.05.2007)
[2]http://www.rfidupdate.com/articles/index.php?id=1354 (Visited 02.05.2007)
[3]http://europe.nokia.com/A4307094 (Visited 02.05.2007)

- Development of supporting infrastructure to design, install, and maintain RFID systems.

3.2 Comparison of technologies

In the previous section three different technologies which can be used to store information about an object or for identifying a given object were presented. The purpose of this section is to find which one is best suited as a physical service discovery technology, in a ubiquitous computing environment. To assess their suitability a set of criteria which is considered to give a good overview of benefits and limitations for application in UbiCollab has been selected. The relative merits of the different technologies for each of the criteria have been summarized in Table 3.2.

Table 3.2: Comparison of AIDC technologies

	Technologies		
Characteristic	**IrDA**	**Optical code**	**RFID**
Ubiquity	High	High	Medium
Data R/W	Yes	No	Yes
Data capacity	High	1-D Low, 2-D Medium	Low - High (price)
Reading distance	Medium	Low	Low - High (technology)
Reading speed	Medium	Low	High (technology)
Requires line of sight	Yes	Yes	No
Robustness	Medium	1-D Medium, 2-D Low	High
Problematic materials	None	None	Metal
"Bulk reading"	No	No	Yes (some readers)
Standardized systems	High	High	Medium
R & D	Low	Low	High
Human visibility	"Red eye" visible	Visible	Hidden
Human readability	None	Sometimes (symbol+text)	None
Tag cost	High	Low	Medium
Reader cost	Medium	1-D Low / 2-D Medium	High

As we can see from the table the different technologies have strong and weak sides. There is no single characteristic that stand out as much more important than others, and as such a technology must be selected from an overall evaluation. The most important aspect which must be considered is that UbiCollab requires a technology which is user-centered and versatile, because of the need to adapt to different application areas.

One characteristic that favours RFID is that it does not require line of sight. IrDA and barcodes on the other side require that the reader is aligned with the object to be read, and is also more sensitive with regard to reading distance. This is important when it comes down to user-friendliness and how much effort the end-user will need to accomplish various tasks. Optically readable codes have to be scanned deliberately by a person and this process is difficult to automate. RFID tags, on the other hand, can also be readily scanned automatically without human involvement.

With regard to the cost associated with the introduction of one of these systems, optical codes are the clear winner. Optical readers are relatively cheap and labels cost almost nothing. If an optical symbol like QR-Code is selected, the cameras integrated in most modern mobile phones would do the job and the software required to create labels is free. IrDA devices are also included in almost any mobile phone, but the price of a large scale deployment of IrDA beacons or tags would be quite high. Deployment of RFID is expensive and that is often considered the biggest problem associated with the technology.

Another aspect in favour of RFID is that RFID tags can be hidden from view and can be placed in protective wrapping which extends both the lifetime and robustness. Barcodes on the other hand are very visible, which sometimes can be desirable but not always. Barcodes are usually created on paper and as such they are sensitive and can easily be destroyed. IrDA beacons/tags must also be visible to be able to connect to them, and will also boast a considerable physical presence. In addition IrDA beacons/tags will require a power supply, which is perhaps the biggest problem with this technology.

When it comes down to which is the most ubiquitous technology today barcodes is the winner because of its use in retail businesses and for tracking goods[21]. Barcodes however is not a technology with much innovation and expansion. It is commonly assumed that RFID with its superior characteristics will replace barcodes as RFID technology continuously becomes cheaper [14] [8]. RFID is already in use in more distinct application areas than barcodes, and continues to create and find its way into new application areas.

Because of its versatility and high level of user-friendliness RFID is considered to be the technology that first and foremost should be supported in UbiCollab. RFID is also considered to be the most future-oriented of the different AIDC-technologies, as it continues to replace older AIDC technology and in other cases creates new applications. Characteristics like non-contact read and write capabilities, high data storage capacity, and hostile environment functionality are also desirable.

A future implementation of QR-code support should also be considered, because of the ubiquity of mobile phones equipped with cameras. Since most modern phones are able to support this technology it would be relatively cheap to use this technology in large scale. In some situations the QR-code could be a nice supplement to RFID, with its high data capacity and relatively high degree of user friendliness.

Chapter 4

Solution proposal

"*The first deadly sin is to code before you think.*"
- Peter J. Brown

This chapter will give a description of the proposed solution for service management in UbiCollab, and presents the design and overall functionality of the chosen platform components. This proposal includes the full service API for the platform components, and also describes how the functional requirements are being mapped to specific components. Design of components for testing and demonstrating the functionality of the solution are also described.

Since a Service Oriented Architecture (SOA) is being used the importance of maintaining loose coupling between services (components) is stressed in the solution. This is particularly evident in the service discovery subsystem, which uses a pluggable solution. In order to provide the full functionality of the service discovery subsystem, several loosely coupled services (components) are joined together to create a composite service. The loose coupling allows services to join and leave at runtime to add or remove functionality.

An important aspect of UbiCollab is that advanced end-users should be able to use the system without spending too much time on training. To achieve this a high level of usability is required. In this context usability refers to the elegance and clarity with witch the end-user interacts with the system. This implies that components for testing and demonstrating the functionality must be designed according to principles from the Human-Computer Interaction (HCI) domain, as these are the components the end-user interacts directly with. Some important principles from the HCI domain that will be used in the solution are presented in Appendix D.

In Section 4.1 important concepts from UbiCollab and the solution proposal is explained. Some important specifications are also stated here.

Then, in Section 4.2 the functionality and APIs for the platform components are described. Also the design of components for testing and demonstration is presented. In addition, a mapping between the requirement specification and the individual components is provided.

Finally, in Section 4.3 the relation of this solution to the overall UbiCollab architecture is described.

4.1 Concepts

In this section a brief description of important UbiCollab concepts will be presented. These concepts provide an introduction to important aspects of UbiCollab and will be commonly referred to in the reminder of this report. Both UbiCollab platform and deployment concepts will be described, as well as solution specific concepts. A more complete description of the platform and deployment concepts are given in the UbiCollab White paper [5].

4.1.1 UbiCollab platform concepts

In this section concepts needed to describe the basic mechanisms and functionality in UbiCollab are presented.

Services

In UbiCollab a *Service* can be a device like an X10[1] lamp controller, a projector or some other resource which is external to UbiCollab. Services constitute resources that can readily be drawn upon to support collaboration in UbiCollab.

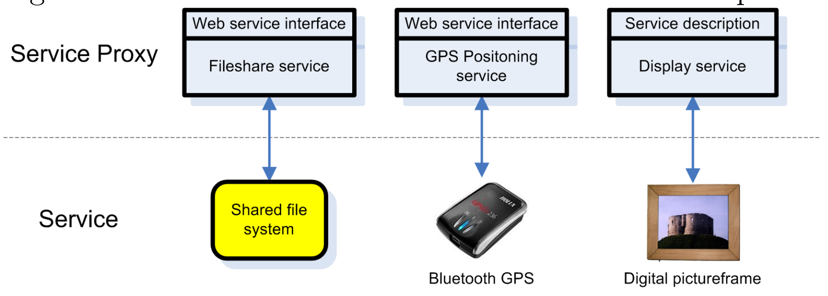
Service Proxies

In order to connect to services and use them in UbiCollab, *Service Proxies* are used. Service proxies are used to hide the underlaying implementation and offers a uniform interface for applications which use the UbiCollab platform. Web-service (WS) technology has been selected for service proxies in order to provide a platform independent solution. Service proxies can be compared with hardware drivers, but instead of providing an operating system dependent interface, a platform independent WS interface is provided. The relation between Services and Service Proxies are visualized in Figure 4.1.

Since the container selected in UbiCollab is OSGi, service proxies will be deployed as OSGi bundles. OSGi bundles are more closely described in the autumn report. Where nothing else is specified, it is assumed that one bundle contains one service.

[1]Wireless and wired power line communication technology, used for lighting and small appliance control

Figure 4.1: The relation between services and service proxies



Service Domain

In UbiCollab end-users will perform many of the tasks which commonly are assumed to be handled by *service providers*. To aid the user in this task the concept of *service domain* is introduced. Every user in UbiCollab has one service domain, and this is the administrative domain where his services are located. The user can install and manage services in his service domain.

Spaces and Collaboration Spaces

The concept *space* in UbiCollab is used to describe a physical space, and is represented using geometric coordinates. Users can define UbiCollab spaces to represent physical spaces which they frequently visit, such as home, office, etc. When a space is shared with other users and thereby used as a resource for collaboration, it is called a *collaboration space.*

Collaboration Instance

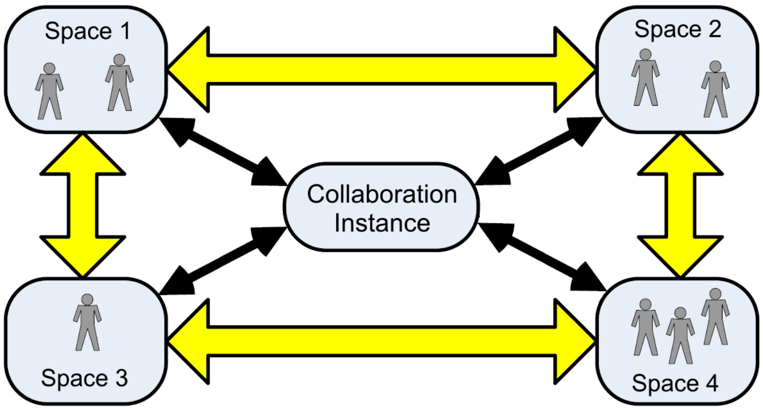
A *collaboration instance* (CI) can be seen as the virtual domain in which collaboration takes place. Resources like *service proxies* and *collaboration spaces* are published to the CI. Information about its participants, collaboration spaces, published service proxies and a generic repository is contained in the CI. This is likely the most interesting concept in UbiCollab for application developers which intend to use the platform.

Human Grid

This is considered to be the basic concept in UbiCollab and is used to describe how people and their artifacts/resources connect together through the use of the UbiCollab platform. Users reside in *spaces* (e.g home, office) which can contain resources such as physical devices, artifacts and various services. These resources constitute the mean for communicating and collaborating with other users in the grid. Collaboration can happen either directly or through a *collaboration instance* which then becomes the virtual context for collaboration.

An example of a human grid is displayed in Figure 4.2. As can be seen from the figure, the human grid ties together many of the other concepts and puts them into context.

Figure 4.2: UbiCollab basic concept: The Human Grid



4.1.2 UbiCollab deployment concepts

In this section some important deployment concepts in UbiCollab will be defined. These concepts are important in order to describe where and how the different components in the solution proposal will be deployed.

UbiNode

An UbiNode is a networked device belonging to a UbiCollab user, running a subset of the main (core/basic) UbiCollab components. The SOA approach allows a UbiNode to retain partial functionality even if not all the main services are running on it. All *service proxies* in a user's *service domain* runs within a UbiNode. Currently, UbiCollab is limited to one UbiNode for each user identity. A UbiNode will typically be a Pocket PC or a laptop computer.

UbiNetwork

In UbiCollab all users sharing the same user management infrastructure constitute what is called a UbiNetwork. A UbiNetwork can normally be identified by a service named “User Manager” which provides centralized user management and trust, but can also exist without one. In the latter case the UbiNetwork will rely on distributed trust, typically provided by a reputation service. For users roaming between UbiNetworks, federation is supported.

UbiHome

A UbiHome node is used to host services common to users located within a UbiNetwork. The shared services in a UbiNetwork, like “User Manager”, will be running on one UbiHome. A UbiHome is typically operated by a dedicated UbiNetwork user (which can be a person or an organization).

4.1.3 Solution specific concepts

One of the problems when working with services and service discovery is that there is a lack of generic domain independent concepts for describing services. This problem comes from the different service technologies which use different terms and concepts. Service related concepts used in the proposed solution will be described in this section.

Service provider

This is the entity (person or organization) which is offering one or more services. Examples of well known service providers on the Internet are Amazon, Telenor and SAS. In UbiCollab every user can be a *service provider*. Every user has a *service domain* where he can host and manage *service proxies*. When he publishes these service proxies to a *collaboration instance*, he becomes a service provider. A service provider can also be a commercial actor, charging for the use of his services.

Service Requester and Client request

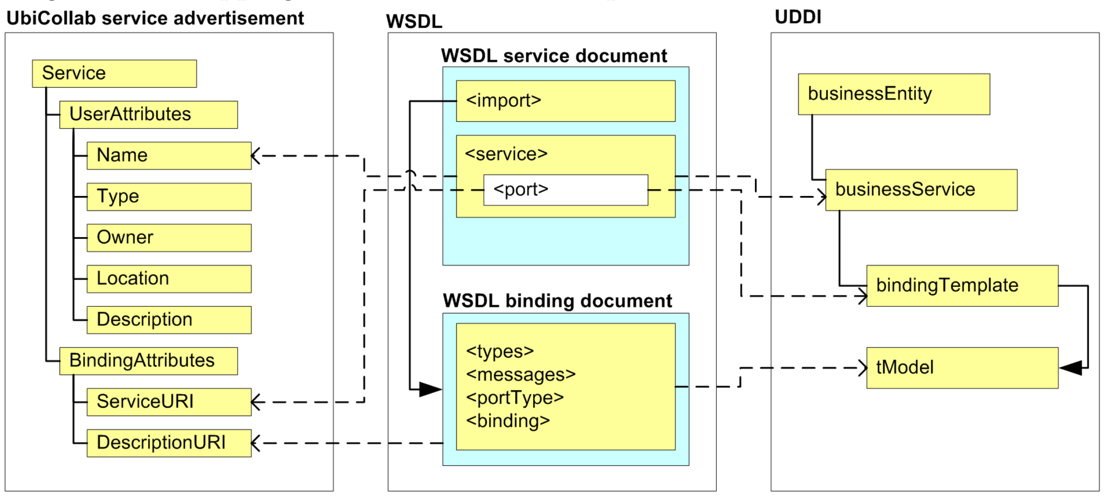
When an entity (computer agent, person or organization) is trying to find a service, he will register a *client request* with the service discovery subsystem. The client request is one of the central concepts in this solution. When an entity registers a client request with the service discovery subsystem, this entity then becomes a *service requester*.

Service description

Service description (SD) for UbiCollab *service proxies* is the Web Service Definition Language (WSDL) file associated with the service. SD is being used somewhat differently by different service discovery protocols. Commonly, SD is used to describe the functional properties of the services. For instance WSDL-files mainly describe the binding information (e.g. methods and parameters) needed to use the service.

As pointed out in the autumn report this is information of little value to the end-user, which is more likely interested in non-functional properties (attributes) such as location, owner, name and a textual description of the service. This information is added to the service description when the service is published. A conceptual mapping between service description and service advertisement for WSDL documents is visualized in Figure 4.3.

Figure 4.3: Mapping between service description and advertisement for WSDL



Service advertisement

The service advertisement (SA) is one of the most central concepts for the service discovery solution in UbiCollab. SA is the descriptive information concerning a service that is made available for potential *service requesters*. When a user registers a *client request*, this is matched against available *service advertisements* in the service discovery process.

Normally the SA is published by the *service provider*, or is implicitly available for self advertising services like UPnP and Bluetooth. The SA contents varies between protocols, but it is usually the *service description* (or a link to where it can be found) and some additional non-functional information.

In UbiCollab the focus is to create a user-centered solution and therefore attributes of value to the user need to be included in the service advertisement. When a user registers a *client request*, it is these attributes that will be used in the service discovery process to match found services with the *client request*. The following attributes have been selected for use:

Name The name of the service, e.g. “X10 Proxy”.

Location The location of the service, e.g. “004, IT-V, ITBygg”. See Chapter 7.3.

Description A short user-friendly description the service.

Owner The owner or responsible of the service. E.g. “UbiCollab project”.

Type The *service type*. Can be used by applications to determine how to handle discovered services. Users or agents can also utilize this attribute in *client requests*, to select which services they are interested in.

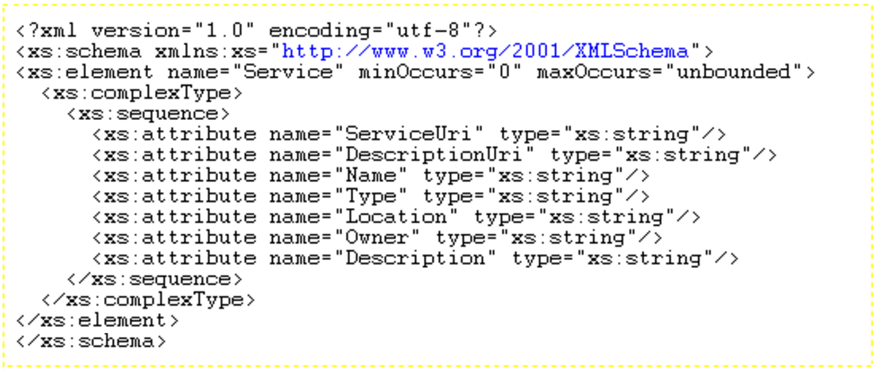
In addition to the user-centered attributes, two attributes will be needed for applications or agents to be able to use the service:

ServiceURI This is the URI that points to where the service can be contacted on the Internet. For a Web-service this will be the address to the web-service, for an information page (HTML) this will be the URL to that page. Also the service address provided by protocols like Bluetooth can be expressed using a URI representation.

DescriptionURI Service types like Web-services and UPnP provide a downloadable service description, in a XML-document. For service protocols which offer this feature, this attribute should be used to point to it.

As UbiCollab seek to be platform independent, XML will be used to describe services. Using XML also has the benefit that additional elements or attributes can be added in future work, and still interoperate with the current platform components, unless the new elements/attributes are mandatory. Since service information will be used and interchanged among several platform components, tools and applications, it is important that they conform to a strict XML Schema Definition (XSD). The *service advertisement* for describing a single service should conform to the XSD shown in Figure 4.4.

Figure 4.4: XML Schema for description of a single service



```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="Service" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:attribute name="ServiceUri" type="xs:string"/>
      <xs:attribute name="DescriptionUri" type="xs:string"/>
      <xs:attribute name="Name" type="xs:string"/>
      <xs:attribute name="Type" type="xs:string"/>
      <xs:attribute name="Location" type="xs:string"/>
      <xs:attribute name="Owner" type="xs:string"/>
      <xs:attribute name="Description" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Note that this XSD is conceptual and does not contain a root-element, which would be required by many XML parsers in order to marshal/unmarshal the XML. When a single service is marshalled to XML, a root element “Service” must be added. If it is a list of services, a root element named “Servicelist” must be used. An XSD for a a list of services is displayed in Figure 4.5.

Figure 4.5: XML Schema for description of a list of services

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="Servicelist">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Service" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:attribute name="ServiceUri" type="xs:string"/>
            <xs:attribute name="DescriptionUri" type="xs:string"/>
            <xs:attribute name="Name" type="xs:string"/>
            <xs:attribute name="Type" type="xs:string"/>
            <xs:attribute name="Location" type="xs:string"/>
            <xs:attribute name="Owner" type="xs:string"/>
            <xs:attribute name="Description" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Service type

The *service type* is an attribute used in the *service description* to describe what kind of service it is. No complete service type hierarchy has yet been defined for UbiCollab, and this will be subject of future work. For the purpose of this work and demonstration purposes, the following types have been defined:

Ubi:Core A downloadable UbiCollab Core component. For instance the “Tag Manager” is not a mandatory UbiCollab core component, but can provide value for a UbiCollab user.

Ubi:Proxy A downloadable UbiCollab Service Proxy.

Ubi:Tag A tag read from an RFID tagged service. The service name attribute will then hold the unique tag identifier.

UPnP A UPnP service.

WS A Web-service.

Info An information page, typically a regular Web-page.

The *service type* attribute can be used by applications to determine how to handle discovered services. This attribute can also be specified in *client requests* by *service requesters*, to select which services they are interested in.

4.2 Functionality and APIs

In this section the mapping between functionality and the components that will be implemented is given. The relation between components is also described, when applicable. For components that provide an application programming interface

(API), this will also be documented. This section is further divided in subsections where components implementing related functionality are described.

In the design of these components the architectural choices from the autumn report will be followed to a large extent. Upon selecting the components to develop especially the design tactic known as “Separate the user interface from the rest of the application” has been followed. This is particularly important since UbiCollab uses a SOA.

By separating the user interface from the remaining functionality, several advantages are gained. The main advantage is the possibility of several user interfaces toward the components. This is highly important, since the components should be possible to run on several kinds of computing devices (E.g. mobile phones, PDAs, laptops, PCs). In addition, other interaction mechanisms like voice commands or physical interfaces can be used. By using this tactic the modifiability tactic of “semantic coherence” will also be strengthened. The “semantic coherence” tactic is also used to a large extent to ensure modifiability and portability.

4.2.1 Service Domain Manager

Service Domain Manager will provide the functionality needed by end-users and/or agents to manage proxy services in the users service domain, located on one UbiNode. This component will be implemented as a Web-service running on OSGi, packaged in an OSGi bundle. Detailed functional requirements assigned to this component are described in Section 4.2.6. The basic operations provided are:

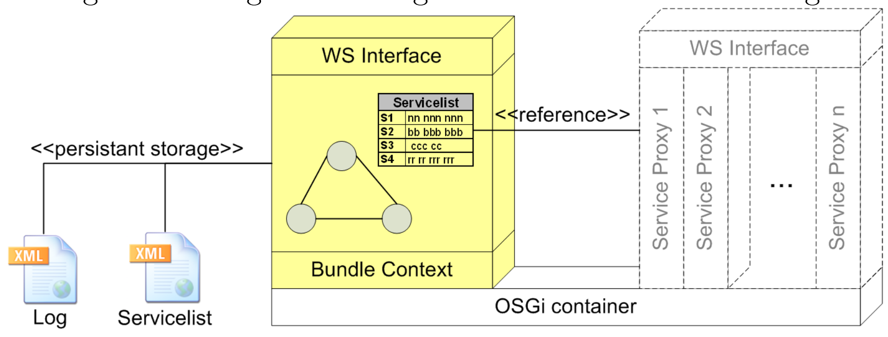
- Install services
- Uninstall services
- Start/Stop services
- Edit service properties
- Retrieve the registered information about a service, or all services
- Retrieve the Service Domain Manager log

In order for the component to get access to other services (bundles) on the OSGi framework, the *BundleContext*[1] interface provided by the framework will be used. This interface either supports directly, or provides the mean to get access to the specific functionality that is needed.

A high level design showing which interfaces the component will need to support, is visualized in Figure 4.6. As can be seen from the figure, the component maintains a list of proxy services installed on the UbiNode (OSGi container), and provide management functionality for these through a Web-service interface.

[1]http://www2.osgi.org/javadoc/r4/ Visited (09.06.2007)

Figure 4.6: High-level design for Service Domain Manager



It is also required that tasks related to management must be logged, and this log made available to the end-user. Therefore a log of all events of interest to the user, ranging from information events to errors and warnings, must be created. This log will be written to an XML file, to enable easy readout when requested by a client. The XML log provided to a client should conform to the XML Schema (XSD) displayed in Figure 4.7.

Figure 4.7: Service domain manager

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="Log">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Logitem" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:attribute name="Date" type="xs:int"/>
            <xs:attribute name="Time" type="xs:string"/>
            <xs:attribute name="Level" type="levelType"/>
            <xs:attribute name="Message" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:simpleType name="levelType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="DEBUG"/>
    <xs:enumeration value="INFO"/>
    <xs:enumeration value="WARNING"/>
    <xs:enumeration value="ERROR"/>
  </xs:restriction>
</xs:simpleType>
</xs:schema>
```

Service Domain Manager will also need to store the list of services persistently, and this persistence should be updated on all changes. Since this list will only be used internally by the component, the format for this list does not need to be defined strictly. However, it should be XML-based to enable portability between different operating systems upon which the UbiCollab platform can run.

As part of the service description for proxy services installed in the service domain, properties describing the space the service is located in must be included. This is necessary to enable support for starting and stopping of services based on space. The properties used to describe the services should be based on UbiCollab service descriptions (Section 4.1.3), but with some additional properties:

BundleID This is the handle needed to gain access to the bundle hosting the proxy service (or other services).

Space This is the textual description of the space the service is located in. UbiCollab default space description can be overridden to provide additional detail, e.g. "Coffee table room 004".

SpaceID This is the ID referring to a specific space, as provided by UbiCollab Space Manager.

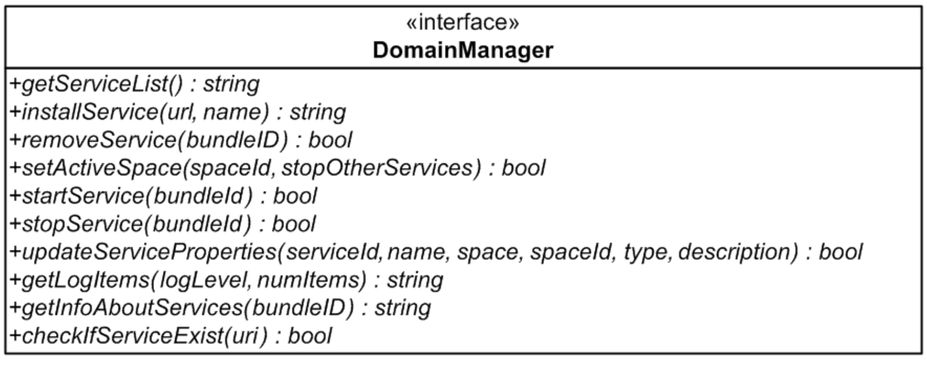
State This is the current state of the service. Can be one of: "Installed", "Uninstalled", "Starting", "Stopping", "Resolved", "Running" and "Stopped".

The solution for the UbiCollab Service Domain Manager should be created general enough to manage all types of services installed on an OSGi platform. Still, it contains additional functionality for *service proxies* and management of these is the primary application area for this component.

4.2.1.1 Service Domain Manager API

The API that will be implemented in and provided by the Service Domain Manager has been created and is displayed in Figure 4.8. Since the component is created as a Web-service, the methods will be accessible through SOAP invocations for clients. The component will itself be responsible for exposing the corresponding WSDL document.

Figure 4.8: API provided by Service Domain Manager



```
«interface»
DomainManager
+getServiceList() : string
+installService(url, name) : string
+removeService(bundleID) : bool
+setActiveSpace(spaceId, stopOtherServices) : bool
+startService(bundleId) : bool
+stopService(bundleId) : bool
+updateServiceProperties(serviceId, name, space, spaceId, type, description) : bool
+getLogItems(logLevel, numItems) : string
+getInfoAboutServices(bundleID) : string
+checkIfServiceExist(uri) : bool
```

A full description of the API with all parameters have been created. The description can be found in Appendix C.1.

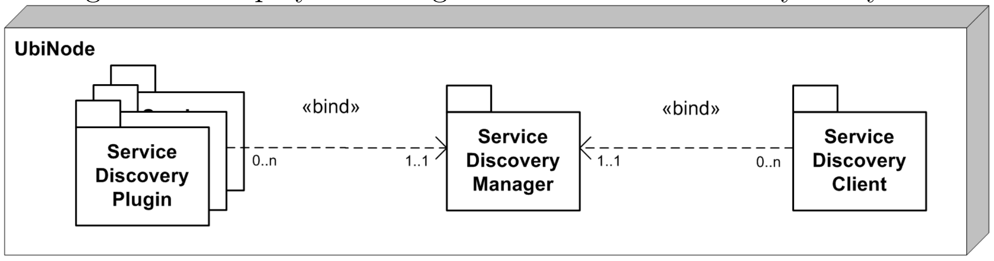
4.2.2 Service Discovery Subsystem

The Service Discovery Subsystem in UbiCollab consists of two distinctly different types of components. These components are named respectively "Service Discovery Manager" (Discovery Manager) and "Service Discovery Plugin" (SDP, or simply plugin). Each SDP will be implementing exactly one service discovery protocol and will provide support for this protocol to the Discovery Manager. Discovery Manager is responsible for mediating client requests and discovered services between clients of the service discovery subsystem and SDP's.

Discovery Manager is the central component in the service discovery subsystem. Support for different service discovery protocols can be added or removed at runtime by adding or removing SDPs. This is one of the benefits of using SOA and loosely coupled components. In order to provide support for the service discovery process both Discovery Manager and at least one plugin is needed, and as such Service Discovery Subsystem is a composite service.

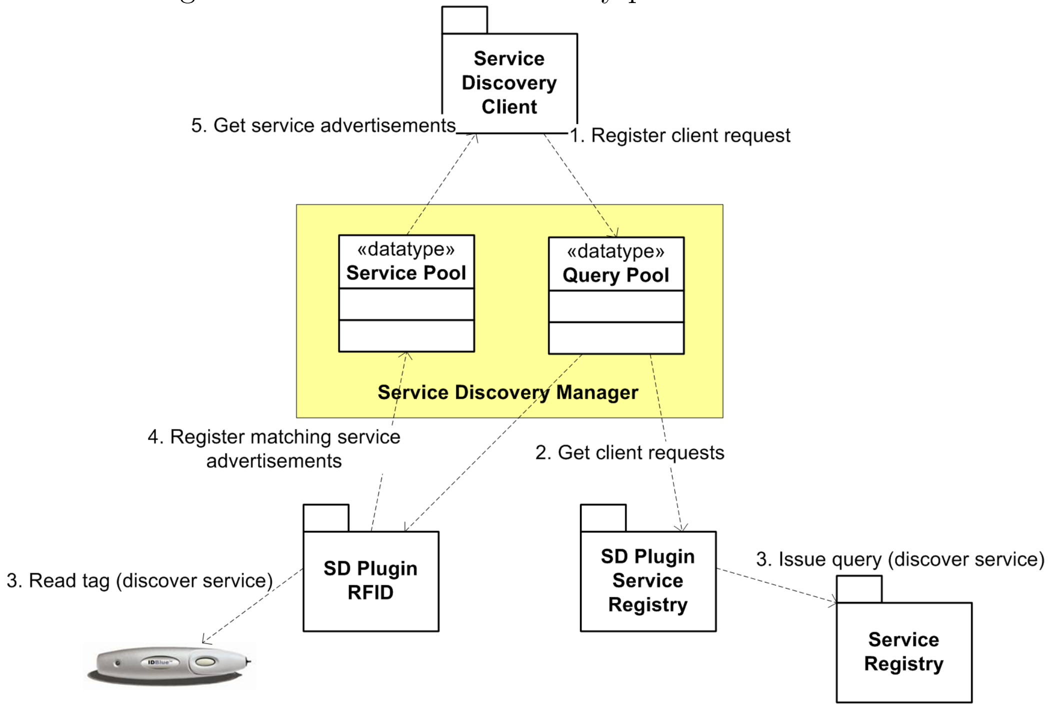
Components in Service Discovery Subsystem are preferably implemented as OSGi bundles and deployed on UbiNodes as visualized in the deployment diagram in Figure 4.9. As the figure shows there can be 0..n SDP's connecting to one Discovery Manager. There can also be 0..n service discovery clients connected to the Discovery Manager at any time. Discovery Manager will provide two API's, one for SDP's and one for clients.

Figure 4.9: Deployment diagram for Service Discovery subsystem



A conceptual design which describes how the different components will support the service discovery process in UbiCollab is shown in Figure 4.10. In this figure only the two SDPs which are planned in this solution proposal is displayed. A prerequisite before the process illustrated in the figure can start, is that both clients and plugins must be registered with the Discovery Manager.

Figure 4.10: The service discovery process in UbiCollab


Service Discovery Client
5. Get service advertisements
1. Register client request
«datatype» Service Pool
«datatype» Query Pool
Service Discovery Manager
4. Register matching service advertisements
2. Get client requests
3. Read tag (discover service)
SD Plugin RFID
SD Plugin Service Registry
3. Issue query (discover service)
Service Registry

As the figure shows, the service discovery process in UbiCollab consists of the following steps:

1. The client registers a request for a specific service with the Discovery Manager. This request is placed in each plugin's request queue.

2. Every plugin will check if any new requests have arrived in its queue, and will get the request.

3. Every plugin will try to use its service discovery mechanism and search for services matching the request.

4. Any service advertisements matching the request will be returned to the Discovery Manager by the plugin's. In the figure only the RFID plugin has found a matching service advertisement.

5. The client will check if any services have been discovered and fetch these.

Which functional requirements are assigned to the different components in the service discovery subsystem is described in Section 4.2.6. The reminder of this section will look at the different components that are part of the subsystem. The rest of this section will be divided into the following subsections:

The **Service Discovery Manager** subsection will be used to describe the functionality and design for this component.

In the **Service Discovery Manager API** subsection the APIs provided by the Discovery Manager will be described.

Then, in the **Service Discovery Plugins** subsection the generic functionality and aspects of plugins are described. Also the impact of some aspects of physical discovery is described here.

Next, in the **Service Discovery Plugin for RFID** subsection the specific functionality and issues that must be handled as part of the RFID plugin implementation will be described.

Finally, in the **Service Discovery Plugin for Service Registry** subsection the functionality and important aspects for the Service Registry plugin will be described.

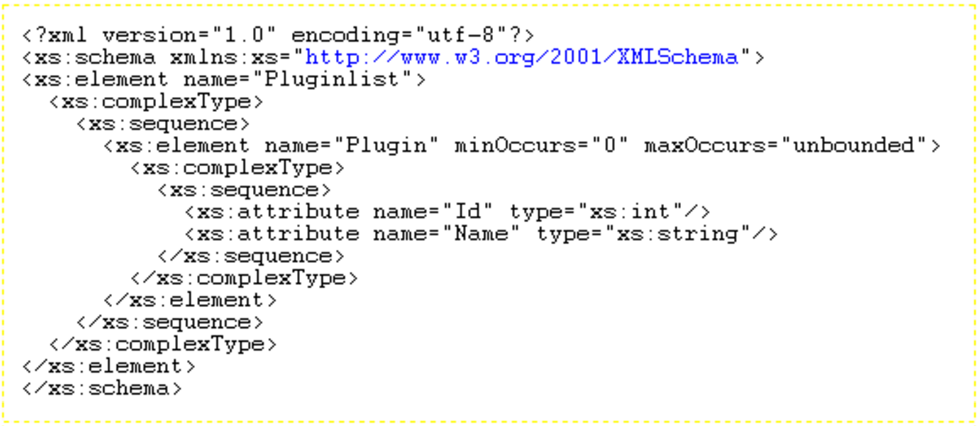
4.2.2.1 Service Discovery Manager

The Discovery Manager is the "hub" in the service discovery subsystem, and sits between clients (users, applications or agents) and SDPs (plugins). It will provide the functionality needed for clients to register requests for services (client requests), and to return matching services. To give the client as much freedom in the selection process as possible, all the high-level properties defined in the service advertisement (Section 4.1.3) can be specified in the client request. The searchable properties are:

- Name - The service name
- Type - The service type
- Location - The location of the service
- Owner - The owner of the service
- Description - The service description

When service advertisements are returned, the format of these will validate to the XML-Schema defined in Figure 4.5. The Discovery Manager also offers functionality that allows clients to get a list of the registered plugins. This will allow the client to know which service discovery protocols are supported. In most cases the client is unlikely to be interested in this information, but for advanced users and debugging purposes this is valuable information. An XML-Schema that defines the format for this list is displayed in Figure 4.11.

Figure 4.11: The XML-Schema for the list of registered plugins

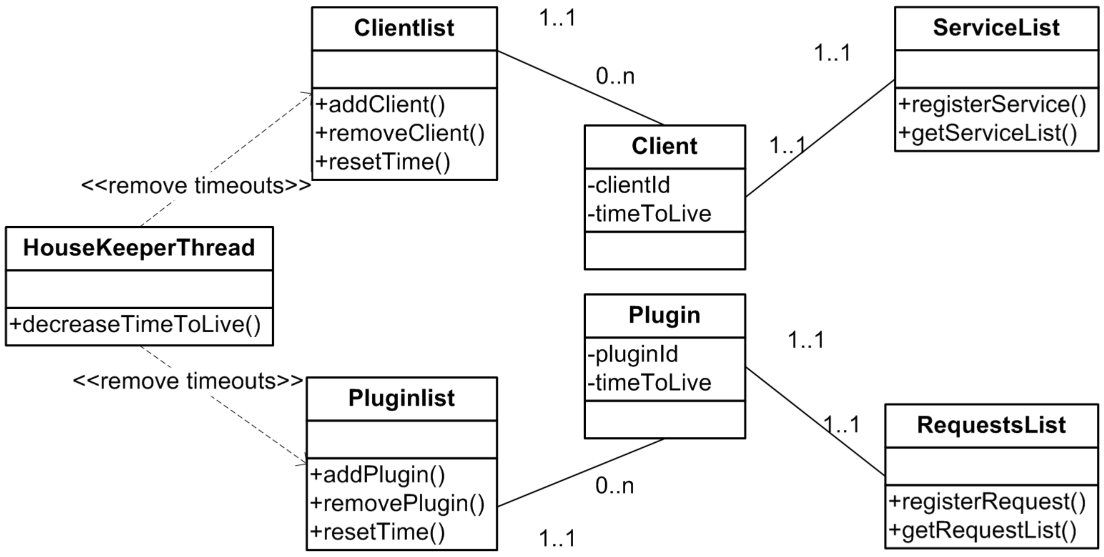


```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="Pluginlist">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Plugin" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:attribute name="Id" type="xs:int"/>
            <xs:attribute name="Name" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Perhaps the most difficult issue when designing this component is to maintain a loose coupling with clients and plugins. In order to maintain a loose coupling the principle of timeout will be used. A completely stateless solution is difficult if not impossible to achieve, but the use of timeout will allow the coupling to be as loose as possible. Both SDPs and client will have to register themselves with the Discovery Manager and will then get an ID which is used to identify them. If too much time passes the Discovery Manager will remove them. Then, the SDP or client will have to re-register themselves.

This will be solved by giving clients and SDPs a specified time-to-live which is reset every time they perform an operation with the Discovery Manager. A "housekeeping" thread runs in the background and decreases the remaining time-to-live at given intervals. When the the time-to-live has reached 0 for a component, it along with its objects will be destroyed and the component will have to re-register. A high-level class diagram showing most important classes in Discovery Manager is displayed in Figure 4.12.

Figure 4.12: High-level class-disgram for Discovery Manager



When a client registers with Discovery Manager, it is added to the `ClientList` and a `ServiceList` is created for the client. If a client registers a `ClientRequest`, this will be added to every plugins `RequestList`. After a service discovery process has been performed the client will fetch all services in its `ServiceList`, and this will then be empty. The list is a FIFO[1] buffer.

At the time a plugin registers itself with Discovery Manager, it is added to the `PluginList` and a `RequestList` is created for the plugin. When a client has registered a `ClientRequest`, this will be avaliable in the plugin's `RequestList` and can be fetched. This list is also a FIFO buffer. If a `ClientRequest` has been fetched and the request has been resolved, the discovered service advertisements will be registered in the requesting client's `ServiceList`. When this is done, the service discovery process is concluded, as described in Figure 4.10.

The format defined for a list of client requests, is defined in the XML-Schema displayed in Figure 4.13.

[1]First In First Out

Figure 4.13: XML-Schema for `ClientRequestList`

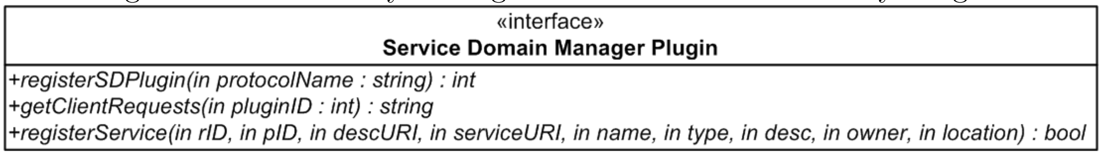
```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="ClientRequestList">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ClientRequest" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:attribute name="ClientID" type="xs:int"/>
            <xs:attribute name="Name" type="xs:string"/>
            <xs:attribute name="Type" type="xs:string"/>
            <xs:attribute name="Location" type="xs:string"/>
            <xs:attribute name="Owner" type="levelType"/>
            <xs:attribute name="Description" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

4.2.2.2 Service Discovery Manager API

As was depicted in Figure 4.12, the Service Discovery Manager will provide two different APIs. One API is needed for Service Discovery Plugins and one is needed for clients. The APIs are displayed in Figure 4.14 and Figure 4.15 respectively. For reasons of space constraint, some parameter names have been abbreviated in the API figures. These names are written in full in the detailed description. The methods provided in each API are described in more details below them.

Since the component is created as a Web-service, the methods will be accessible through SOAP invocations. The component will itself be responsible for exposing the corresponding WSDL document.

Figure 4.14: Discovery Manager API for Service Discovery Plugins

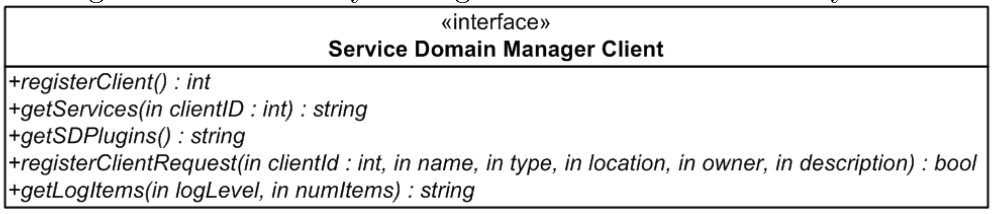


«interface»
Service Domain Manager Plugin

+registerSDPlugin(in protocolName : string) : int
+getClientRequests(in pluginID : int) : string
+registerService(in rID, in pID, in descURI, in serviceURI, in name, in type, in desc, in owner, in location) : bool

The methods provided in the plugin interface are described in more detail in Appendix C.2

Figure 4.15: Discovery Manager API for service discovery clients



«interface»
Service Domain Manager Client

+registerClient() : int
+getServices(in clientID : int) : string
+getSDPlugins() : string
+registerClientRequest(in clientId : int, in name, in type, in location, in owner, in description) : bool
+getLogItems(in logLevel, in numItems) : string

The methods in the client API are described in Appendix C.3

4.2.2.3 Service Discovery Plugins

Services supported by UbiCollab span a range of different service types like Webservices, UPnP and Bluetooth. In addition to these traditional service types, UbiCollab expands the definition of what services are to also include other types of resources. Examples of such resources are the physical discovery of RFID tags which can be resolved to complete service advertisements. These service advertisements can then advertise for instance service proxies which can be downloaded and installed, or simply information pages.

This section will look at the generic properties and unique aspects of different types of services, and describe a solution to how these can be supported. As described in the autumn report, service discovery protocols can be grouped according to their scoping. The two common categories to use are:

Multicast-based These protocols principally exposes their services through the use of “self-advertisement”. Common examples include Bluetooth, UPnP and WS-Discovery.

Centralized registries This type of services can be available across the Internet, or can be limited (e.g. to a company LAN/WAN).

These two categories of protocols have two common characteristics which allow them to be handled the same way when implemented as plugins. These characteristics are:

- They are network based.
- The discovery process for the respective protocols can be initiated by a computer application or agent. It can also be initiated by a user, but this will then be mediated by some application.

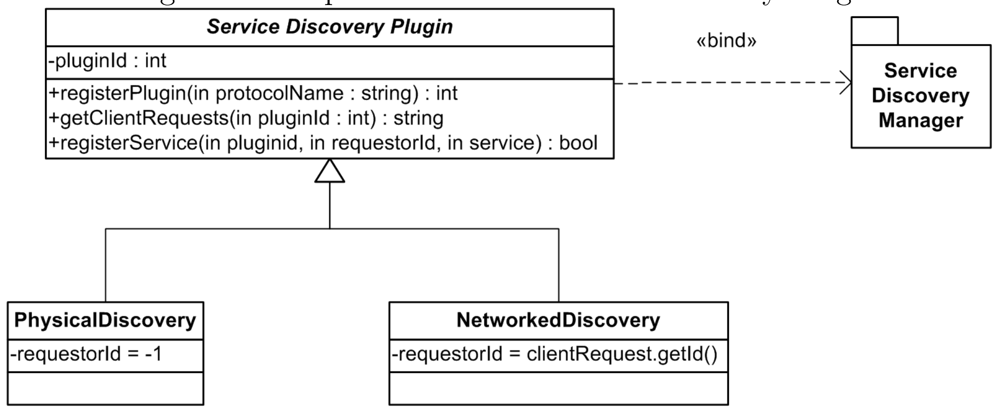
Physical discovery on the other hand does not necessarily comply with these characteristics. Perhaps the most important aspect with physical discovery is that it is usually user initiated by some physical gesture. The consequence for discovery is that the user specifies by pointing, rather than through descriptive parameters, what he is looking for.

The implication of this is that two specializations of a generic Service Discovery Plugin is needed. Figure 4.16 shows the Service Discovery Plugins inheritance hierarchy. The two specializations of the abstract class `Service Discovery Plugin` comes from the way client requests for services are initiated.

The `NetworkedDiscovery` class will retrieve client requests from Service Discovery Manager, try to resolve these and return matching service advertisements. Plugins of class `PhysicalDiscovery` will discard client requests, and instead return all service advertisements found as the result of physical discovery.

As the figure shows, the problem with addressing of the returned service advertisement for `PhysicalDiscovery` plugins will be solved by setting `requesterId = -1`.

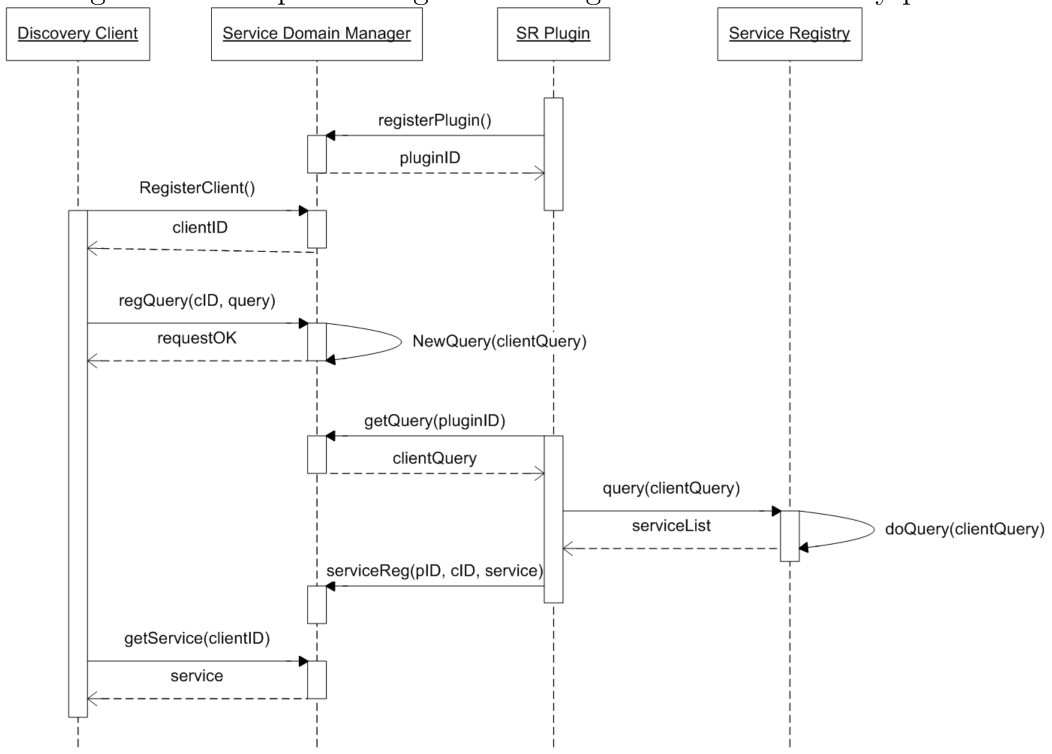
Figure 4.16: Specialization of Service Discovery Plugin


Service Discovery Plugin
-pluginId : int
+registerPlugin(in protocolName : string) : int
+getClientRequests(in pluginId : int) : string
+registerService(in pluginid, in requestorId, in service) : bool
«bind»
Service Discovery Manager
PhysicalDiscovery
-requestorId = -1
NetworkedDiscovery
-requestorId = clientRequest.getId()

When Discovery Manager receives a service advertisement registration with `requesterId = -1` this advertisement is added to the service list for registered clients.

The principle where clients and plugins register with the Discovery Manager and a simple service discovery process is performed, is visualized in Figure 4.17. In this diagram the Service Discovery Plugin for Service Registry is used as example for the plugin. This component is more closely described in Section 4.2.2.5.

Figure 4.17: Sequence diagram showing the service discovery process

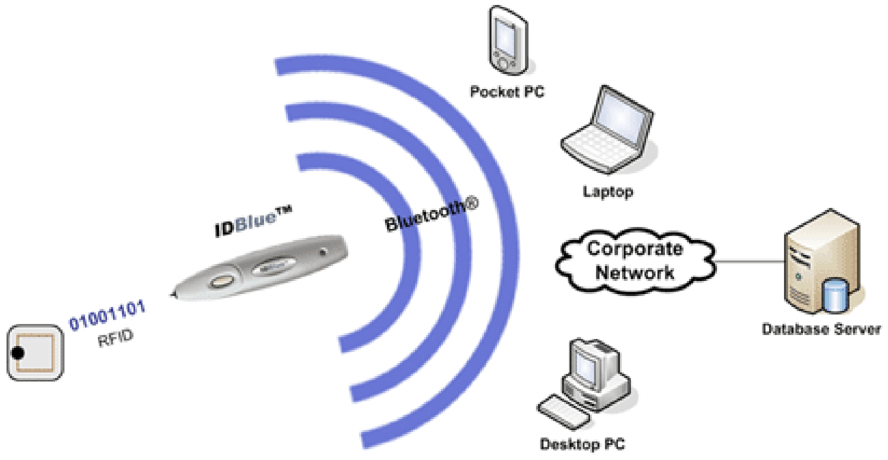


As can be seen from the figure, the client's identifier is issued when the client registered with the Discovery Manager. It follows the client request through the whole discovery process. That is, all the way until the discovered advertisements are registered back to the Discovery Manager and placed in the queue (service list) belonging to the correct client. Note that the plugin's identifier is also registered along with the service advertisement. This allows the client to know which discovery protocol (plugin) has returned the advertisement.

4.2.2.4 Service Discovery Plugin for RFID

The RFID device selected for use in the service discovery solution is named "ID-Blue" and is produced by Cathexis Innovations[1]. This is a Bluetooth RFID reader in the shape of a pen. An overview of the technical details of interest for this solution have been compiled, and is available as Appendix E. Possible applications for the reader is displayed in Figure 4.18. The figure shows the pen being pointed toward an RFID tag in order to read it.

Figure 4.18: Possible uses of IDBlue RFID reader



Examples of devices that can connect to the pen using Bluetooth (Pocket PC, Laptop computer, Desktop PC) are also displayed in the figure. Different usage scenarios require different solutions. For instance the pen can be used in offline mode (not connected to a master Bluetooth device) and read up to a thousand tags, which can be transmitted to e.g. a desktop PC when connected.

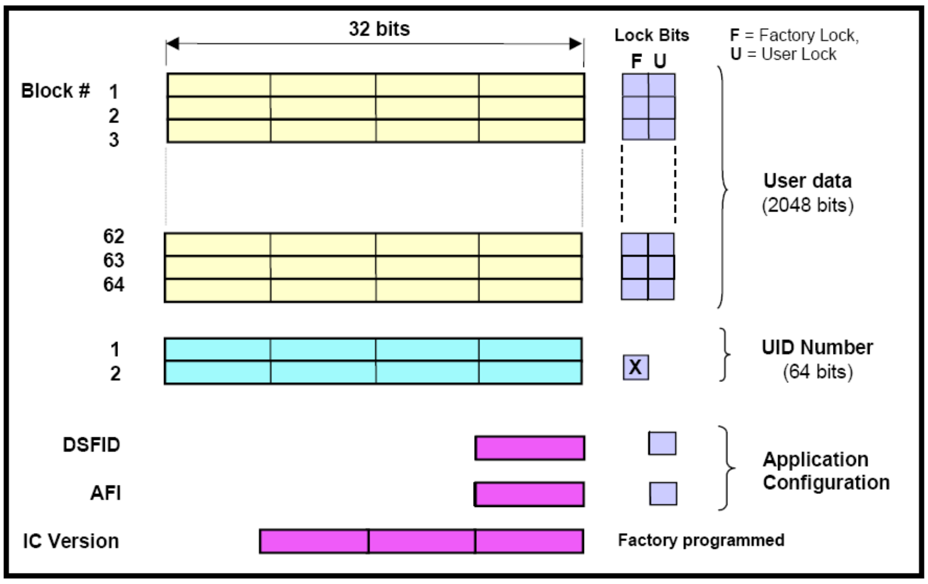
The IDBlue device supports two fundamental operating modes, synchronous and asynchronous. In synchronous mode commands are streamed directly from the connecting device to the IDBlue reader without any user interaction. This mode is fully software controlled and suitable for operations which take a lot of time, typically writing tags or reading a large amount of data. In asynchronous mode the user triggers simple actions by pressing the button on the IDBlue device.

Asynchronous operations are much faster and more user-interactive than synchronous operations. Asynchronous operations however are limited in the amount of data they can handle. Still, because of the user-centered focus for the solution, the asynchronous mode is the preferred one. This mode allows the user to deliberately point at services in the vicinity and thereby explicitly show his/her interest.

[1]http://www.cathexis.com/ (Visited 10.06.2007)

The IDBlue reader supports three different different RFID standards, and in this solution the RFID tags selected for use are ISO 15693-2 compliant tags. This solution will use tags from the Texas Instrument[1] "Tag-It" series. In Figure 4.19 the organization of data on these tags is visualized. As shown in the figure the tag offers 2048 bits of storage space, which equals 256 bytes or characters provided an 8 bit character set is used. These must be read out in blocks of 4 bytes, and the IDBlue device supports reliable reading of around 12-16 blocks in asynchronous operation.

Figure 4.19: Organizing of data on RFID tag



For encoding of characters the 8-bit ASCII (American Standard Code for Information Interchange) character encoding set[2] has been selected. This encoding scheme provides the characters necessary to encode almost any legal URI, which is what will be encoded on the tags. The escape character selected to end an encoded URI is the space character " ", as this character is illegal in Internet style URIs.

To ensure reliable reading from the tags, a length of 12 blocks will be used. This implies that a maximum of 48 characters will be available for encoding. If this is too short for some uses, an intermediate should be utilized for redirecting. An URI redirecting is simple to make and in addition there are many free solutions for this availiable on the Internet. One possibility could be to use a service like TinyURL.com[3] to map a short URL e.g. "http://tinyurl.com/6" to the real URL which can have an arbitrary length.

The URI encoded in the tag must point to an XML-file with service advertisements. This allows the RFID plugin to be free from dependencies resulting from the limited amount of data that can be read. In addition, the RFID solution

[1]http://www.ti.com/rfid/ (Visited 10.06.2006)
[2]http://www.unicode.org/charts/PDF/U0000.pdf (Visited 10.06.2007)
[3]http://tinyurl.com (Visited 05.06.2007)

can then be used for service discovery without relying on components other than Discovery Manager. The XML-format in this file must comply with the XML-Schema in Figure 4.5. There also exists another option for tagging which does not require the encoding of a URI. This possibility relies on the existence of a Service Registry, and will map the unique tag identifier to one or more service advertisements in the registry. The Service Discovery Plugin will support both these options.

In addition to this, the plugin needs a way to connect to the IDBlue device. The device uses the Bluetooth Serial Port Profile (SPP) over Bluetooth Generic Access Profile (GAP) and as such can be connected to as a Bluetooth "emulated" serial port. This however would force the user to install the device as a Bluetooth "emulated" serial port, and connect to it before every use in an OS-level interface (E.g. Windows Bluetooth devices). The IDBlue.NET developer guide[1] recommends against this.

In order to create a user-friendly solution, the plugin will use a direct Bluetooth connection to connect to the device. This implies that Bluetooth discovery should be implemented in order to find the correct device to connect to. This also allows the use of different IDBlue devices. The Franson Bluetools libraries[2] will be used for this.

To ensure a user-friendly solution this plugin will need a GUI to support these mechanisms. The GUI should be created according to the guidelines provided in Appendix D (HCI principles for arranging GUIs). In addition, the IDBlue support libraries are only available for the .NET platform[3] (both full and compact framework), therefore this will be the platform of choice for this component.

4.2.2.5 Service Discovery Plugin for Service Registry

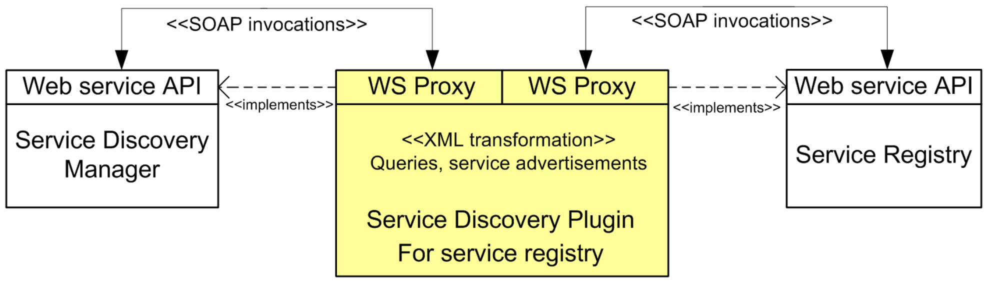
This component will provide search capability in the centralized Service Registry described in Section 4.2.3, for the Service Discovery Subsystem. The API for plugins provided by the Discovery Manager, as well as the API provided by the Service Registry component will be used. This component will be an intermediate between the Service Registry and the Discovery Manager.

A high level design showing which interfaces the component will need to support, is visualized in Figure 4.20. As can be seen from the figure, the component's core functionality is to be an intermediate and transforms messages between Service Registry and Discovery Manager. The component will do this by fetching client requests from Discovery Manager, try to resolve them with Service Registry and then register potential found service advertisements with Discovery Manager.

The component will utilize a thread to time when client requests will be fetched from Discovery Manager. This is done in order to ensure that the principle of

[1]Electronically available as part of the IDBlue package.
[2]http://franson.com/bluetools/ (Visited 10.06.2007)
[3]http://msdn2.microsoft.com/en-us/netframework/ (Visited 10.06.2007)

Figure 4.20: High-level design for Service Discovery Plugin for Service Registry



loose coupling is maintained. When started the thread will try to register the component with Discovery Manager, and if successful will start to query for client requests at given intervals. If disconnected or a Discovery Manager is not found the thread will sleep for 30 seconds and then try to reconnect.

4.2.3 Service Registry

Service Registry will be the centralized service registry in UbiCollab, normally deployed on an UbiHome node. This component will provide basic functionality similar to UDDI, but will be a much simpler component. Detailed functional requirements assigned to this component are described in Section 4.2.6. To enable configuration, testing and management of this component, a complimentary UbiTool[1] will also be created as an independent component. The UbiTool will be named “Service Registry Management Tool”. The basic operations provided by these components will allow users to:

- Register service advertisements
- Remove service advertisements
- Query the registry for service advertisements
- Edit service advertisement properties
- Retrieve the Service Registry log

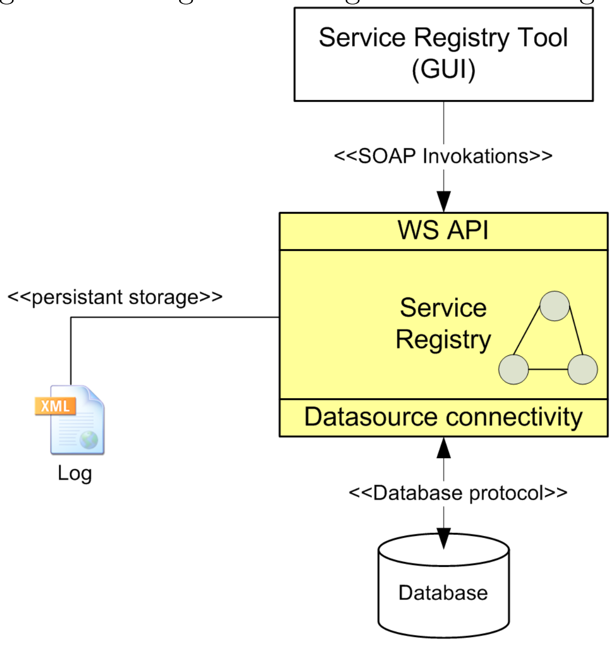
A high level design showing the interfaces the Service Registry will need to support, is visualized in Figure 4.21. The component's core functionality is to receive and resolve queries for service advertisements, and to provide management functionality for these through a Web-service interface.

As the figure shows, a database is used to store service advertisements. The database used for this solution will be the MySQL Server[2]. The quality tactic described in the autumn report named “Maintain semantic coherence” will be used when implementing the DB interface component. This will allow this component to be easily replaced by another component if the database choice changes (e.g. a simple XML file or other database).

[1]Small UbiCollab application interacting with platform components

[2]http://www.mysql.org/ (Visited 10.06.2007)

Figure 4.21: High-level design for Service Registry

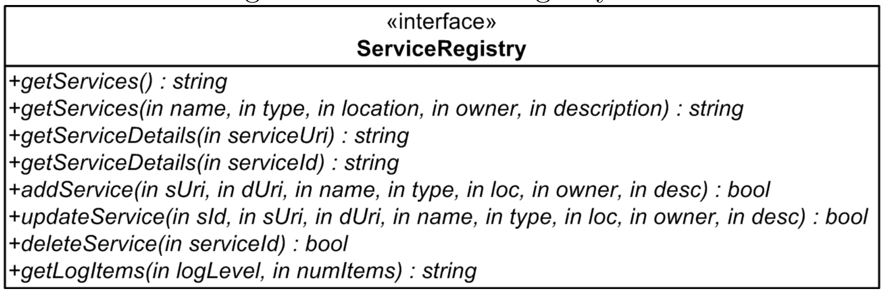


To achieve a user-friendly solution, the "Service Registry Management Tool" will be created according to the guidelines provided in Appendix D. The .NET platform[1] will be used to avoid using to much time on the development of this component. Visual Studio and .NET is regarded superior to most Java IDE (Integrated Development Environments) with regard to GUI design.

4.2.3.1 Service Registry API

The API that will be implemented in and provided by Service Registry has been created and is displayed in Figure 4.24. Since the component is created as a Web-service, the methods will be accessible through SOAP invocations for clients. The component will itself be responsible for exposing the corresponding WSDL document.

Figure 4.22: Service Registry API



```
«interface»
ServiceRegistry
+getServices() : string
+getServices(in name, in type, in location, in owner, in description) : string
+getServiceDetails(in serviceUri) : string
+getServiceDetails(in serviceId) : string
+addService(in sUri, in dUri, in name, in type, in loc, in owner, in desc) : bool
+updateService(in sId, in sUri, in dUri, in name, in type, in loc, in owner, in desc) : bool
+deleteService(in serviceId) : bool
+getLogItems(in logLevel, in numItems) : string
```

Detailed descriptions of the methods in the interface are included in Appendix C.4.

[1]http://msdn2.microsoft.com/en-us/netframework/ (Visited 10.06.2007)

4.2.4 Service Management Tool

Service Management Tool (SMT) will both serve as a demonstrator GUI and as a tool in UbiCollab (UbiTool). It provides end-users with a user-friendly environment where they can find (discover) services, install these in their service domain, and share (publish) these services to make them available in collaboration instances where the users are members.

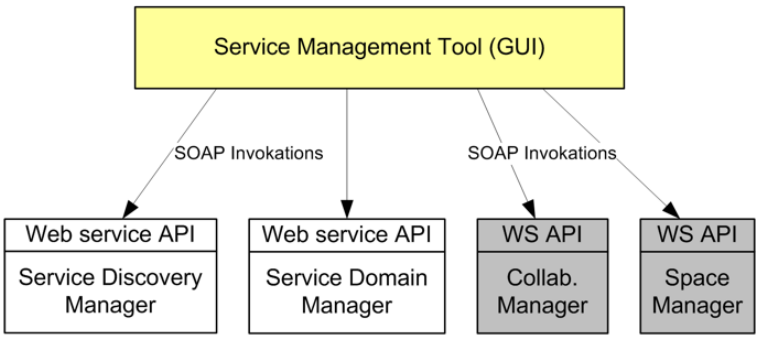
SMT will demonstrate the functionality provided by Service Discovery Subsystem and Domain Manager (components described in sections 4.2.1, 4.2.2.1, 4.2.2.4 and 4.2.2.5). In addition SMT will consume two other UbiCollab platform services named “Space Manager” and “Collaboration Manager”. These components are created in parallel with this work as part of another master thesis [19].

Detailed functional requirements assigned to the component are described in Section 4.2.6. The basic functionality provided by SMT can be summarized to:

- Manage service’s in a users service domain, located on one UbiNode. In addition to support the functionality provided through the Service Domain Manager’s API described in Section 4.2.1.1. Related functionality will be:
  - Get space information from the “Space Manager” component and allow this to be integrated with the service description for service proxies.
  - Support publishing and unpublishing of services to and from Collaboration Instances. The remote API needed for this will be provided by “Collaboration Manager”
- Perform service queries and utilize the API provided by Service Discovery Manager, as described in Section 4.2.2.2. This component must be able to install discovered service proxies in the user’s service domain. In addition it should be possible to view discovered information pages (Web-pages).
- The tool must be able to fetch and display the log from both Service Discovery Manager and Service Domain Manager.

A high level design showing the services this component will consume, is displayed in Figure 4.23.

Figure 4.23: High-level design for Service Management Tool



In the figure, only the components which the Service Management Tool is directly connected to is displayed. The functionality and possibilities provided in the GUI will also rely on other components, e.g. Service Discovery Plugins. How everything ties together is further described in Section 4.3 (Relation to overall UbiCollab architecture).

Because the component relies on several other components and accesses these through SOAP invocations, it should be created as a multi-threaded application in order to avoid delays often associated with Web-services. The following threading structure will be used:

- Main thread: The functionality related to service management will run in the main thread.
- Discovery thread: The functionality related to service discovery processes will run in a separate thread.
- Collaboration thread: The functionality related to Space Manager and Collaboration Manager will run in a separate thread.

The .NET platform[1] will be used to develop this component. The main reason for this is that this UbiTool will contain a rich graphical interface with much functionality, and this is anticipated to be more easily achieved by using Visual Studio and .NET instead of the Java based alternatives. In addition .NET supports the use of delegates for GUI updates in multi-threaded applications, instead of callback functions/methods.

The implementation of this component will also rely heavily on the quality tactics described in Section 7.3.1 (Usability), in the architecture part of the autumn report. Also the quality tactics described regarding performance (Section 7.3.3) will be used in the implementation, in addition to the multi-threading to ensure a "smooth" user experience. The graphical design of the GUI will be based on the principles described in Appendix D.

4.2.5 X10 demonstrator

As mentioned before, one of the important aspects of UbiCollab is the focus to integrate with the physical environment. To demonstrate and evaluate the complete solution, a credible and visible demonstrator service proxy is needed. For this purpose a demonstrator based on the X10 protocol for controlling electrical appliances through the power grid has been chosen. This service proxy will be named "X10Proxy".

This proxy can also be used in further demonstrations of the platform, and as such is expected to continue to provide value for the platform. An accompanying Web-service client that will provide a GUI for the X10Proxy will also be developed.

[1]http://msdn2.microsoft.com/en-us/netframework/ (Visited 10.06.2007)

For this solution an X10 control device named "cm11a" which provides an RS232 serial port interface will be used. Serial connectivity from Java will be achieved through the use of the Java Comm package. The limitations posed by this package, along with a short installation instruction is included in Appendix F.

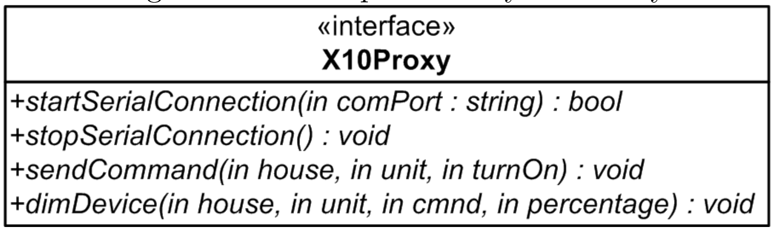
For X10 connectivity the jpeterson library[1], which is available for download as part of the "CruiseControl framework"[2] or directly from the creator's website[3], will be used. This library provides an event-driven model for interacting with the cm11a device. The API for the framework is included in the package.

To ensure a user-friendly solution for demonstration purposes, the GUI component will be created according to the guidelines provided in Appendix D. For a relatively easy and quick development of GUI components, the .NET platform[4] will be used. By combining .NET and Java this way, a fine example of the platform independence provided by UbiCollab is given at the same time.

4.2.5.1 X10 demonstrator API

The API that will be provided by the X10Proxy has been created and is displayed in Figure 4.24. Since the component is created as a Web-service, the methods will be accessible through SOAP invocations for clients. The component will itself be responsible for exposing the corresponding WSDL document.

Figure 4.24: API provided by X10Proxy



```
+startSerialConnection(in comPort : string) : bool
+stopSerialConnection() : void
+sendCommand(in house, in unit, in turnOn) : void
+dimDevice(in house, in unit, in cmnd, in percentage) : void
```

A detailed description of the methods in the interface is given in Appendix C.5.

4.2.6 Requirement matrix

In this section a mapping between the functional requirements specified in Appendix B and the components defined in this chapter has been performed. The purpose of this mapping is to ensure that all requirements are attended to. This is especially important for requirements that span more than one component, in order for the complete required functionality to be implemented.

The resulting mapping has been assembled in a matrix which is displayed in Figure 4.25. The different components are listed in columns, while there is one

[1]Pure Java implementation of the X10 protocol, created by Jesse Peterson
[2]http://cruisecontrol.sourceforge.net/ (Visisted 10.06.2007)
[3]http://www.jpeterson.com/x10/x10.zip (Visisted 10.06.2007)
[4]http://msdn2.microsoft.com/en-us/netframework/ (Visited 10.06.2007)

requirement identifier per row. These identifiers are used to reference the requirements specified in Appendix B.

Figure 4.25: Mapping between functional requirements and components

	Component						
Requirement identifier	**Service Discovery Manager**	**Service Discovery Plugin**	**Service Domain Manager**	**Service Registry**	**Service Mgmt. Tool**	**Other service or tool**	**Comment**
FR-P1						X	Collaboration Instance Manager
FR-P2						X	Collaboration Instance Manager
FR-P3					X		
FR-P5					X		
FR-P6					X		
FR-P7					X		
FR-S1	X						
FR-S2					X		
FR-S3		X		X			
FR-S4	X	X		X			
FR-S5	X	X		X	X		
FR-S6			X				
FR-S7	X				X		
FR-S8	X				X		
FR-S9	X			X			
FR-S10	X	X					
FR-S11	X	X					
FR-D1	X	X	X	X			
FR-D2							Generic requirement
FR-D4			X				
FR-D7			X				
FR-D8			X	X		X	
FR-D9							Generic requirement
FR-D10	X						
FR-D11			X		X		
FR-M1			X				
FR-M2			X				
FR-M3			X				
FR-M4			X				
FR-M5			X				
FR-M6			X				
FR-M7			X				
FR-O1	X	X	X				
FR-O2			X				
FR-O3			X				
FR-O4			X			X	Collaboration Instance Manager
FR-O5					X	X	
FR-O6	X		X	X			
FR-O7						X	Not to be implemented
FR-O8						X	Not to be implemented
FR-O9			X				
FR-O10						X	Not to be implemented
FR-O11	X	X					Partly to be implemented
FR-O12	X		X	X	X	X	Service Registry Tool as well

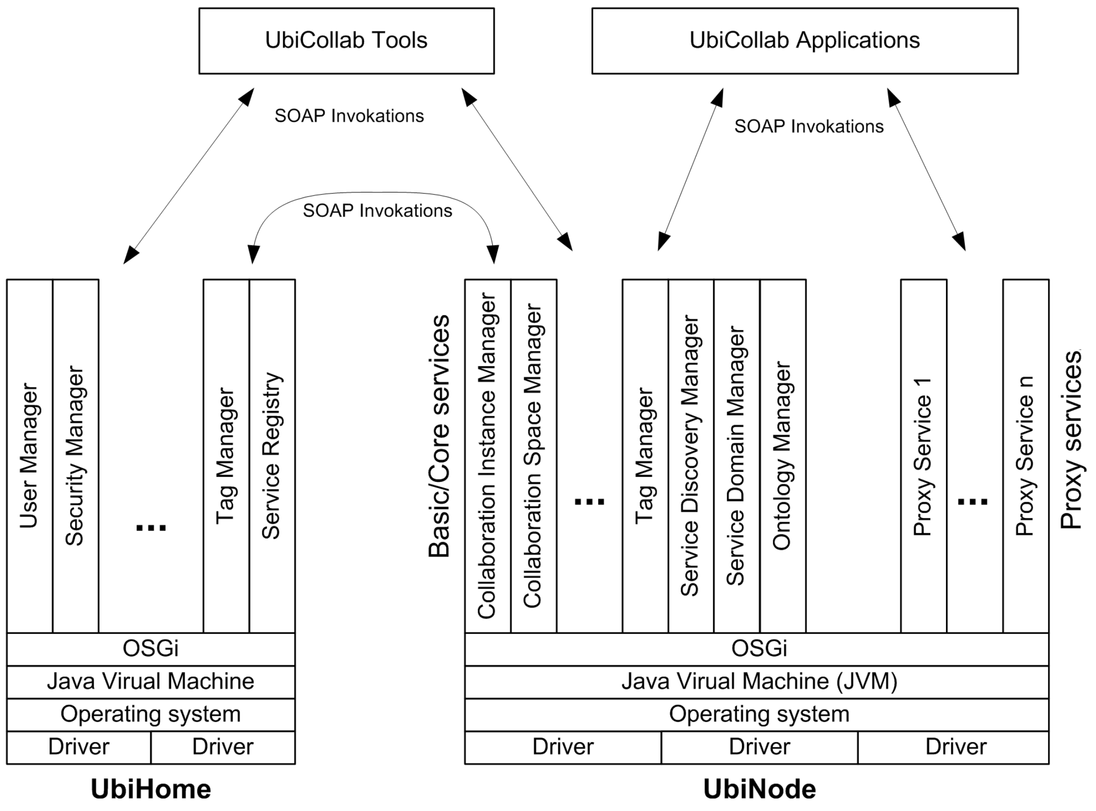
Note that there are some comments attached to certain requirements. When the comment refers to components not included in this work, it will be further explained in Section 4.3. Comments about "Generic requirement" covers all components, while requirements commented not to be implemented will be described further in Section 5.3.

4.3 Relation to overall UbiCollab architecture

In this chapter a description will be given as to how this contribution fits into the overall UbiCollab architecture. A figure which illustrates how the different platform services (bundles) are deployed on UbiNodes and UbiHome, and how they communicate is shown in Figure 4.26. All communication in the figure uses Web-service calls (SOAP invocations) and are principally performed vertically (including communication between services running on UbiHome and UbiNodes). As described earlier this ensures a loose coupling between platform services, and between platform services and clients (UbiTools and applications).

The figure also depicts that Service Registry runs on UbiHome and as such there is normally only one instance of this service in each UbiNetwork. In principle the Service Registry can run on any node, not necessarily UbiHome (however it should run on a node with a high availability). Service Domain Manager and Discovery Manager run on UbiNodes as shown in the figure. Domain Management is necessary for basic services and should run on every UbiNode. Service Discovery Manager is not necessary to have on each node, but in most cases it will greatly enhance the usability.

Figure 4.26: Relation to overall UbiCollab Architecture

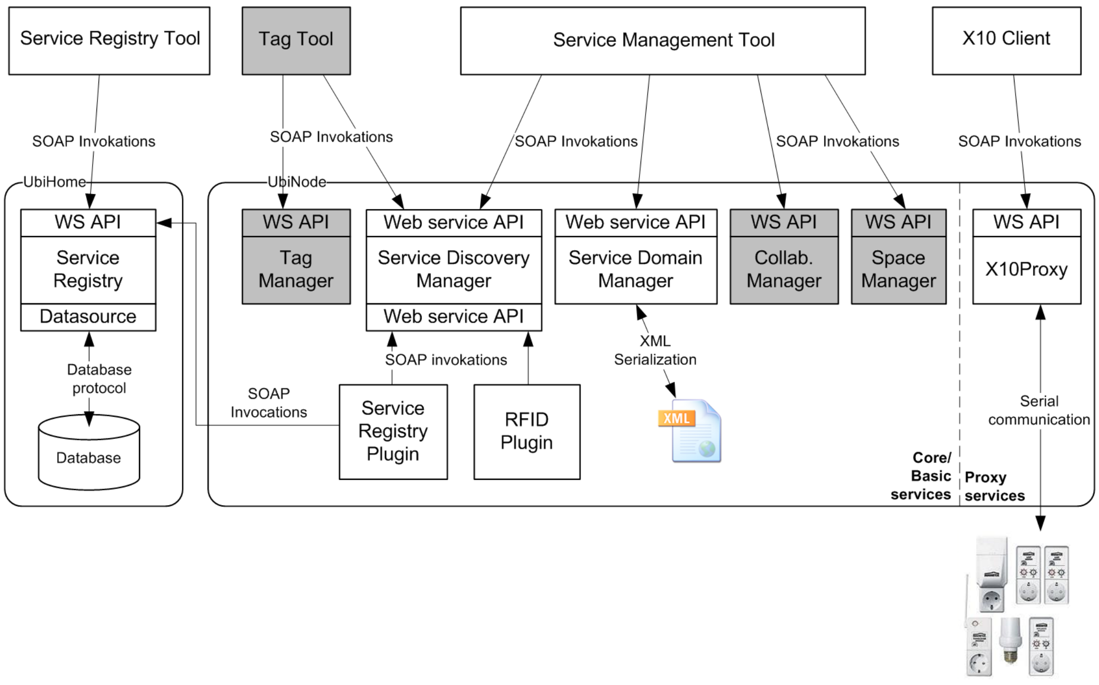


The components created as part of the other master thesises which run in parallel with this, have also been included in the figure. Some of the components like

“User Manager” and “Security Manager” are only conceptual and no implemented components covering these areas have been created as yet.

A more detailed view of the different components that are part of this work, and how they are communicating, is displayed in Figure 4.27. In this figure it is also illustrated how the work performed on Tag management uses the service discovery solution [15]. The work performed in other master thesises is shown in grey. The figure also shows how Service Management Tool uses Space Manager and Collaboration Manager as described in Section 5.2.5.

Figure 4.27: Relation between platform and demonstrator components



In the figure the GUI components have been placed outside of the respective UbiHome and UbiNode depicted. This is only a conceptual placement, as their location is unimportant for the functionality provided by the platform. Since the platform uses Web-service technology these can reside anywhere as long as there is a network connection. Note also how the UbiNode has been divided between basic services and proxy services, which reside in the user’s service domain. It is also illustrated how X10Proxy controls the “real” service which resides outside of the UbiNode.

Chapter 5

Implementation

"In theory there is no difference between theory and practice. In practice there is."
- Yogi Berra

This chapter describes how the solution proposed in Chapter 4 (solution proposal) has been implemented. A short introduction is given where some impotent aspects of the implementation are described. Then, the different component implemented will be described in detail, and the requirements not implemented will be investigated.

Because the platform component implemented are to be published on Sourceforge as part of the UbiCollab open source project, more time have been spent in order to create well-documented and easy-to-read programming code for these components. For the demonstration application and GUI tools, the main focus has been to provide a user interface where the proposed functionality is provided is a user-friendly way. However, also the programming code for these components has been documented.

Section 5.1 explains some aspects of importance to the implementation.

In Section 5.2 the implementation of the components that were designed in the solution proposal will be presented.

Then, in Section 5.3 the requirements that have not been implemented will be pointed out and analyzed.

5.1 Introduction

This section explains some aspects of importance to the implementation. These aspects include documentation of code, specifics of the OSGi framework, problems encountered and limitations to the documentation.

5.1.1 Documentation of code

All code in the implemented components is commented using the native documenting system for the programming language used. The platform components written in Java are extensively documented using JavaDoc[1]. Demonstration and GUI components was written in the C# programming language, and this code are documented using ".NET XML documentation"[2].

5.1.2 OSGi

All platform services except RFID Plugin for Discovery Manager were implemented as OSGi bundles. The Knopflerfish OSGi framework[3] has been selected for UbiCollab use and was used for deployment. Knopflerfish is based on the Java programming language, which means that the bundles were implemented using Java. The services APIs are provided as Web service interfaces by using the "axis-osgi" bundle included in Knopflerfish. When this bundle is installed and started, it can be used to publish service interfaces for bundles as proper Web service interfaces.

Platform services created for the OSGi framework includes a class named `Activator` with a minimum of two methods, named "start()" and "stop()". This class is used by the OSGi framework to start and stop the service provided by the component (OSGi bundle). For all components except Domain Manager, this is the only class that is tightly coupled with the OSGi framework. This solution provides maximum portability, as only this class would need modification if the solution was to be ported to another framework/container.

5.1.3 Problems encountered

Some problems were encountered during the implementation phase and will be documented here. These are mainly related to the use of hardware devices and connectivity with these.

Java Comm API and X10 connectivity

In order to connect to the X10 controller device (cm11a) selected for use in the solution, serial connectivity using RS232 had to be used. To achieve this connectivity the Java implementation for X10 that was selected (jpeterson X10 library[4]),

[1]http://java.sun.com/j2se/javadoc/ (Visited 14.06.2006)
[2]http://msdn2.microsoft.com/en-us/library/aa288481(VS.71).aspx (Visited 14.06.2007)
[3]http://www.knopflerfish.org (Visited 15.06.2007)
[4]http://www.jpeterson.com/x10/x10.zip (Visisted 10.06.2007)

required the Java Comm API. Unfortunately, the Java Comm API for windows has been discontinued[1] for some years and is currently only supported for Linux and Solaris operating systems. The Windows version can still be downloaded from some sources and installed. Possible download sites and installation procedure are described in Appendix F.

IDBlue RFID pen

In order to connect to the IDBlue RFID Pen selected for use as a physical discovery mechanism, several Bluetooth dongles had to be tried before one that worked was found. The following USB dongles was tried in their respective order:

- *DLINK DBT-120* - Did not work. This was the first choice as it was actually recommended by the producer of the IDBlue pen, Cathexis Innovations, on their homepage[2]. However it turned out the dongle need to be "H/W Ver.:B4" or newer, while the one tried had "H/W Ver:B3".
- *Gigabyte GN-BT03D* - Did not work. Sometimes found the IDBue MAC adress but was never able to resolve its name or connect to it. After some e-mails back and fourth to the GigaByte and Cathexis Technical support, it turned out that the GigaByte dongle uses a Bluetooth stack not supported by the IDBlue device.
- *Jensen Scandinavia Blue:Link101* - Worked perfectly and discovered the IDBlue at once.

5.1.4 Limitations

Because of the large implementation which consists of more than 60 class-files distributed over 9 different components, complete class diagrams have not been created for all of these. Class diagrams have been created for platform components which provide an API, and the different classes in these diagrams are described. A complete and detailed description of the APIs is included in Appendix C. Implemented demonstrator and GUI components are documented through screenshots. These are explained along with important aspects of the components.

In the class diagrams some non-essential classes have been removed in order to improve readability and avoid cluttering. Also some non-essential methods like empty constructors and getters/setters have been omitted. In most classes attributes and parameter have also been hidden to improve readability. Components which provide an API display the implemented interface at the top of the diagram. UML format has been used to describe the exported class names (indications) for interfaces.

[1]http://java.sun.com/products/javacomm/ (Visited 15.06.2007)
[2]http://www.cathexis.com (Visited 15.06.2007)

5.2 Implemented components

In this section the implementation for the components designed in the solution proposal (Chapter 4) will be presented. For platform components, UML dependency class diagrams showing the essential classes will be displayed. Demonstrators and tools (GUI components) are described using screenshots from the finished GUI.

In most of the class diagrams the `Log` class has been included. This class is common to all components and provides persistent logging in an XML-format as described in Section 4.2. To provide easy access and ensure at most one instance exist at any time, this class has been implemented using the singleton design pattern.

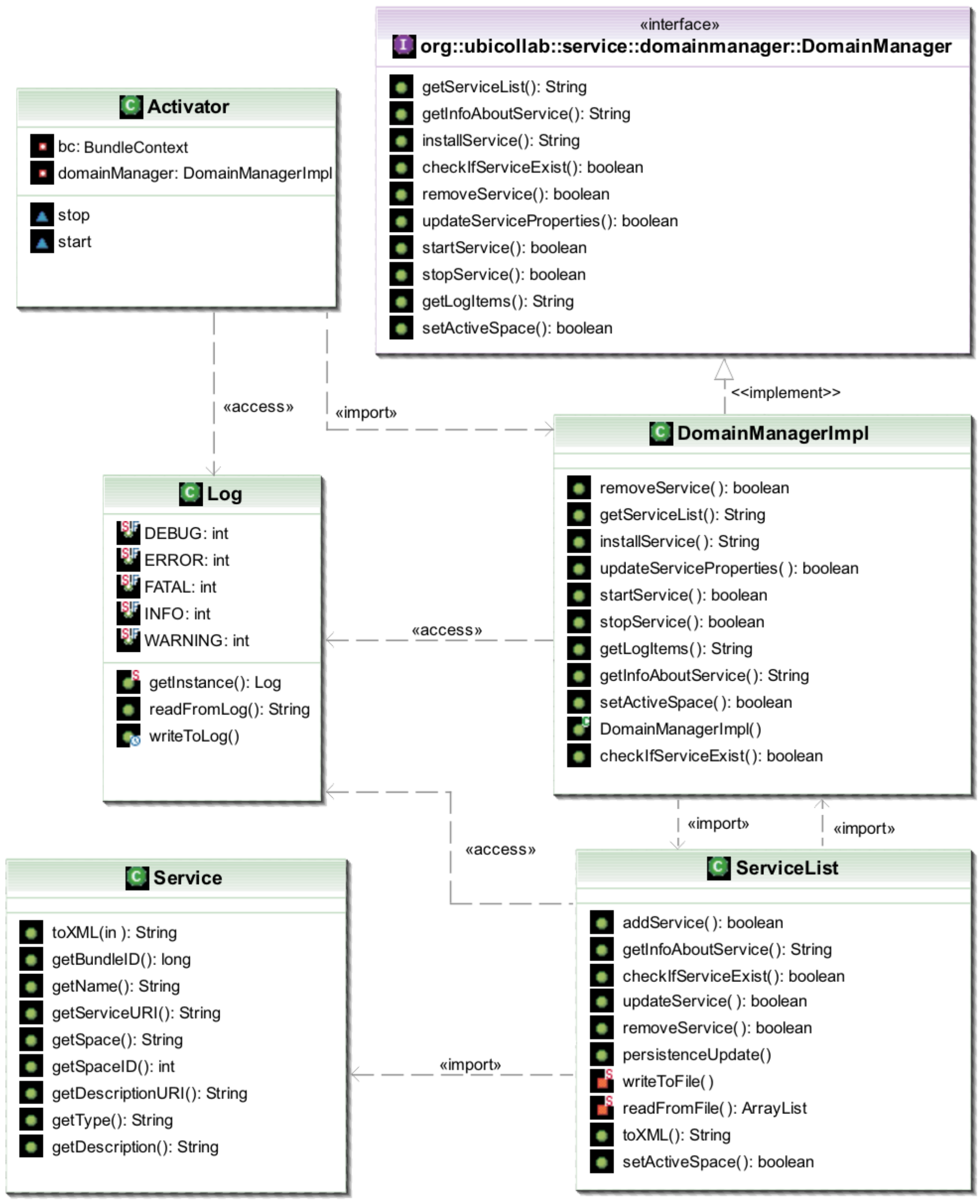
Classes used to represent data structures which at some point can be serialized to XML and returned as the result of some query have a method named `toXML()`. This method has a similar functionality as the inherited `toString()` method. It will return a XML-representation of the object in question.

5.2.1 Service Domain Manager

Service Domain Manager is an UbiCollab platform component implemented in Java, for the OSGi platform. It provides clients with the functionality needed to manage services within the OSGi framework. Therefore this component interfaces with both clients through its provided SOAP interface, and the OSGi framework by means of the `BundleContext` interface provided by the framework. The complete API for this component is described in Appendix C.1.

A class diagram showing the essential classes for Domain Manager is displayed in Figure 5.1. The `DomainManagerImpl` class implements the DomainManager interface, and it is this class which exposes its public methods in the SOAP interface. Those management operations which involve access to or changes to the service list calls methods in the class `ServiceList`. The `ServiceList` class is also responsible for persistent storage and retrieving of the service list. The service list maintains a list of objects, of class `Service`.

Figure 5.1: Domain Manager class diagram



Note that both `Service` and `ServiceList` classes have `toXML()` methods. These methods are used to return respectively an XML-representation for a service or a service list.

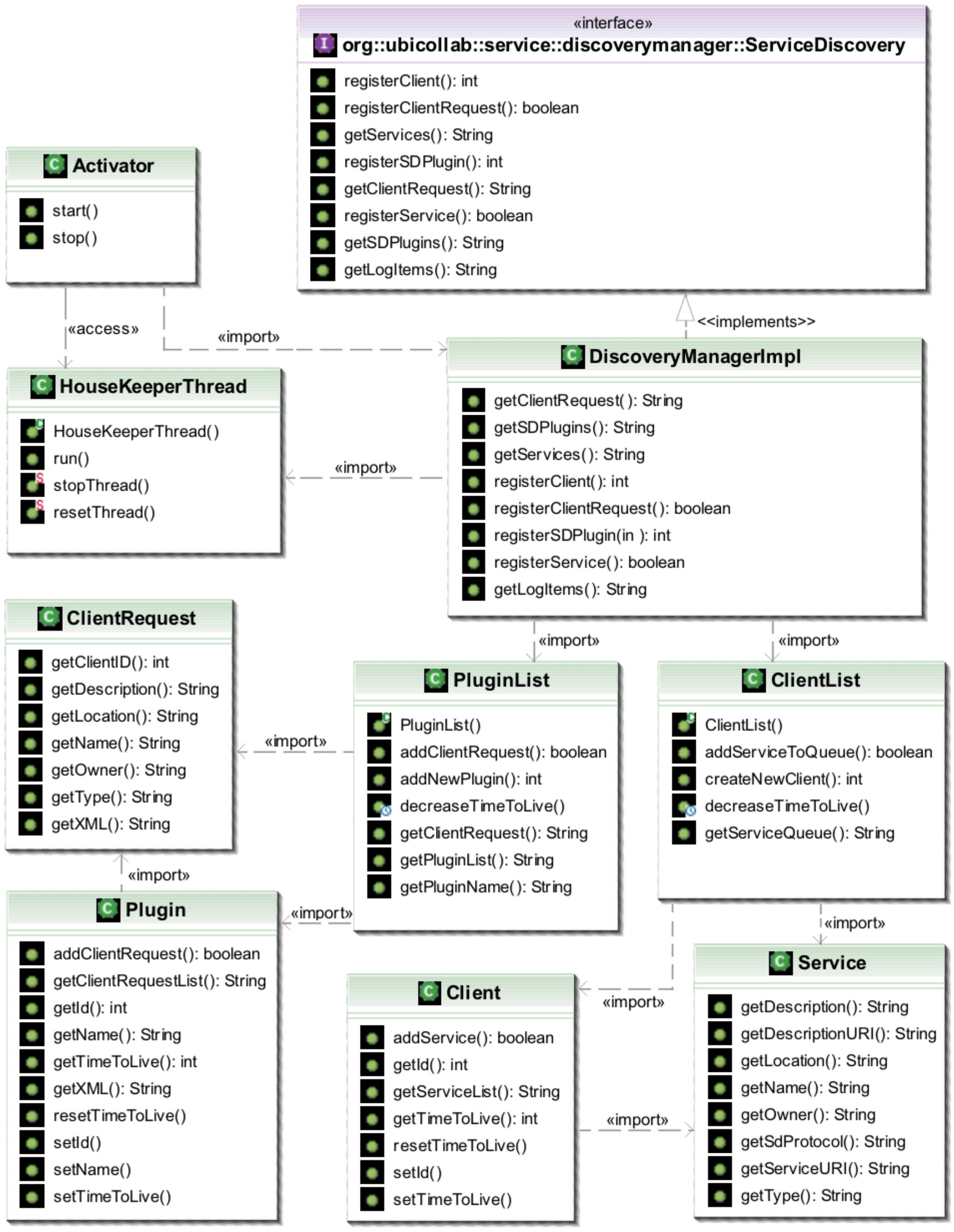
5.2.2 Service Discovery Subsystem

In this section the components which are part of the Service Discovery Subsystem will be explained.

5.2.2.1 Discovery Manager

Service Discovery Manager is an UbiCollab platform component implemented in Java, for the OSGi platform. Discovery Manager functions as a "hub" between service requesters (clients) and service discovery plugins (plugins). To provide this functionality the Discovery Manager offers an API for clients and plugins. The complete API for this component is described in Appendix C. A class diagram showing the essential classes for Discovery Manager is displayed in Figure 5.1.

Figure 5.2: Discovery Manager class diagram



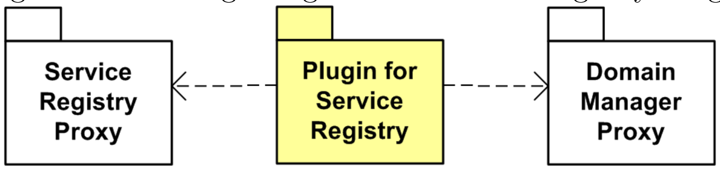
The `DiscoveryManagerImpl` class implements the `DiscoveryManager` interface, and it is this class which exposes its public methods in the SOAP interface. The `HouseKeeperThread` class runs as a thread and is responsible for maintaining a updated `PluginList` and `ClientList`. This is done by keeping track of a time-to-live variable for each plugin and client, represented by the classes `Plugin` and `Client`.

The `ClientList` and `PluginList` classes are used to keep track of registered clients and plugins (respectively Client class objects and Plugin class objects). The `Client` class maintains a list of services found, where each service is of type `Service`, while the `Plugin` class maintains a list of received client requests of type `ClientRequest`.

5.2.2.2 Service Discovery Plugin for Service Registry

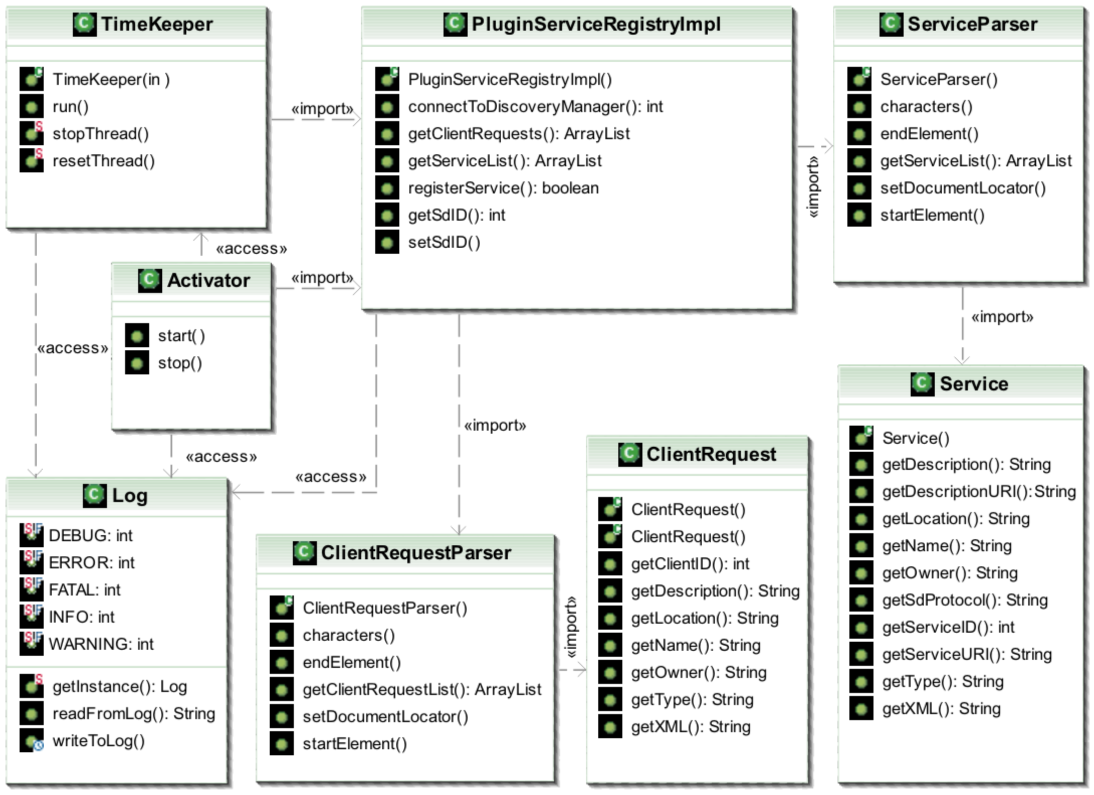
Service Discovery Plugin for Service Registry (SRPlugin) is an UbiCollab platform component implemented in Java for the OSGi platform. It provides search capability in the Service Registry for Service Discovery Manager. This component therefore interfaces with both those components, and this have been done by creating a Java proxy package for each of these. No class diagram will be presented for the proxy packages since these contain "standardized", generated classes. The package diagram for the component is displayed in Figure 5.3.

Figure 5.3: Package diagram for Service Registry Plugin



A class diagram showing the essential classes for the SRPlugin component is displayed in Figure 5.4. The actions performed by the SRPlugin are being coordinated by the class `TimeKeeper`, which runs as a thread. Both the `ServiceParser` and the `ClientRequestParser` classes implements an SAX (event driven) XML parser, to unmarshal respectively `Service` and `ClientRequest` objects from the received XML. Most of the logic in the component is located in the `PluginServiceRegistryImpl` class.

Figure 5.4: Plugin for Service Registry class diagram



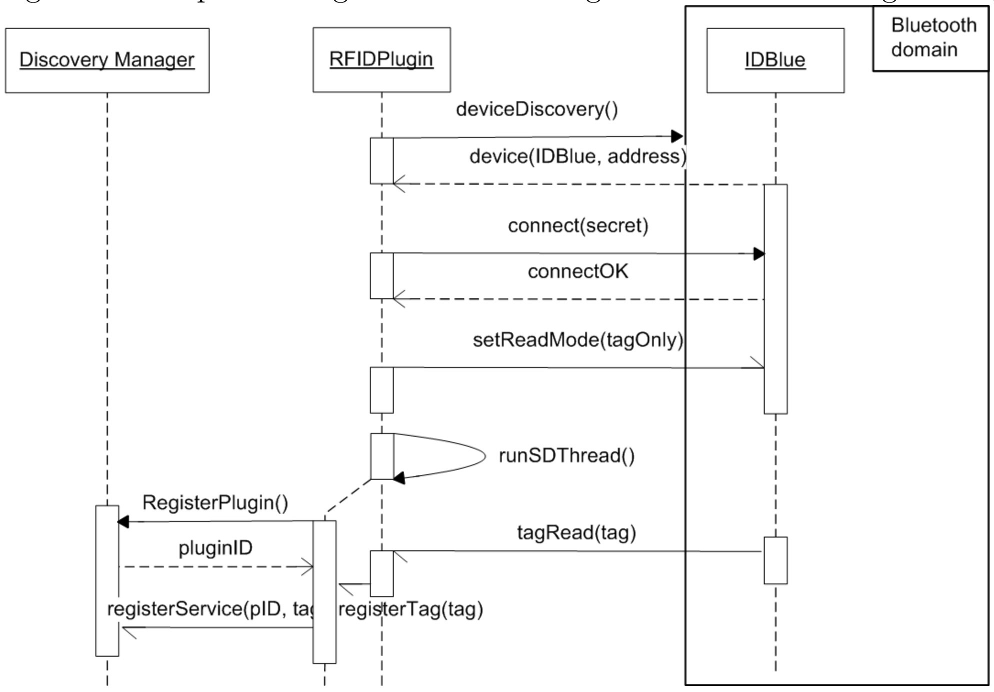
5.2.2.3 Service Discovery Plugin for RFID

Service Discovery Plugin for RFID (RFIDPlugin) is an UbiCollab platform component implemented in C# and running on the .NET platform. It provides physical search capability by means of RFID for the Service Domain Manager. The RFID device supported by this implementation is the Cathexis IDBlue RFID Bluetooth Pen (IDBlue). The development of this plugin has followed the recommended practice as stated in the "IDBlue.NET Developer Guide", bundled with the IDBlue device.

In the implementation the IDBlue.NET API "driver" is created as a singleton object, in order to ensure only one such object exists. The driver runs in a thread of its own and provides an event-driven model, where the plugins main form binds to the events which can be raised by the driver. This provides for a clean implementation and a relatively loose binding. Only the main form in the plugin would need modifications if another device was selected for use.

An example of how a connection to the IDBlue device is opened, and then to the Discovery Manager, is displayed in the sequence diagram in Figure 5.5. First a Bluetooth discovery is initiated and the IDBlue responds. In order to open a connection a shared secret (known as a passkey in Bluetooth jargon) must be provided, as displayed in the figure. The default value for IDBlue devices (as set by the manufacturer) is “0000”, and this value is also used in the implementation.

Figure 5.5: Sequence diagram for RFIDPlugin connection and tag reading



Next, the operating mode for the the IDBlue is default set to “Select Tag ID”. As the diagram shows, when this is done a new thread is started to open a connection with the Discovery Manager (DMThread). The plugin registers itself with the Discovery Manager and gets an identifier. Then, the IDBlue has read a tag and an event is raised in the RFIDPlugin. The data read from the IDBlue (binary string) is interpreted and passed in a message to the DMThread, which again registers the tag as a service advertisement with the Discovery Manager.

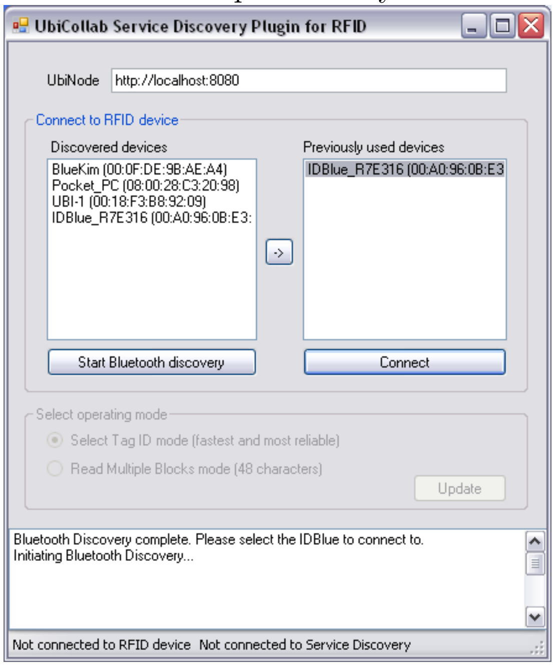
In addition to setting the read mode to “Select Tag ID”, some other settings are set by the plugin during the initial connection. These are:

- RFID Protocol: ISO15693
- Device timeout: 5 minutes
- RFID timeout: 4 seconds

The GUI provided by the RFIDPlugin is displayed in Figure 5.6. At the time the screenshot were taken, a Bluetooth discovery had been completed and the IDBlue was selected but still not connected. Items in “Discovered devices” will when moved to the list named “Previously used devices” be stored in a file and “remembered”, so that they can be connected to without having to perform a Bluetooth discovery every time. In order to connect to a device it must be selected in the list and the “Connect” button must be pressed.

As specified in the solution proposal chapter, two operating modes are supported in the “Select operating mode” control in the GUI. The RFIDPlugin also have

Figure 5.6: The GUI provided by the RFIDPlugin



UbiNode: http://localhost:8080

Discovered devices:
BlueKim (00:0F:DE:9B:AE:A4)
Pocket_PC (08:00:28:C3:20:98)
UBI-1 (00:18:F3:B8:92:09)
IDBlue_R7E316 (00:A0:96:0B:E3:

Previously used devices:
IDBlue_R7E316 (00:A0:96:0B:E3

Select operating mode:
(●) Select Tag ID mode (fastest and most reliable)
( ) Read Multiple Blocks mode (48 characters)

```
Bluetooth Discovery complete. Please select the IDBlue to connect to.
Initiating Bluetooth Discovery...
```

Not connected to RFID device Not connected to Service Discovery

a text area at the bottom of the GUI where events of interest for the user are printed.

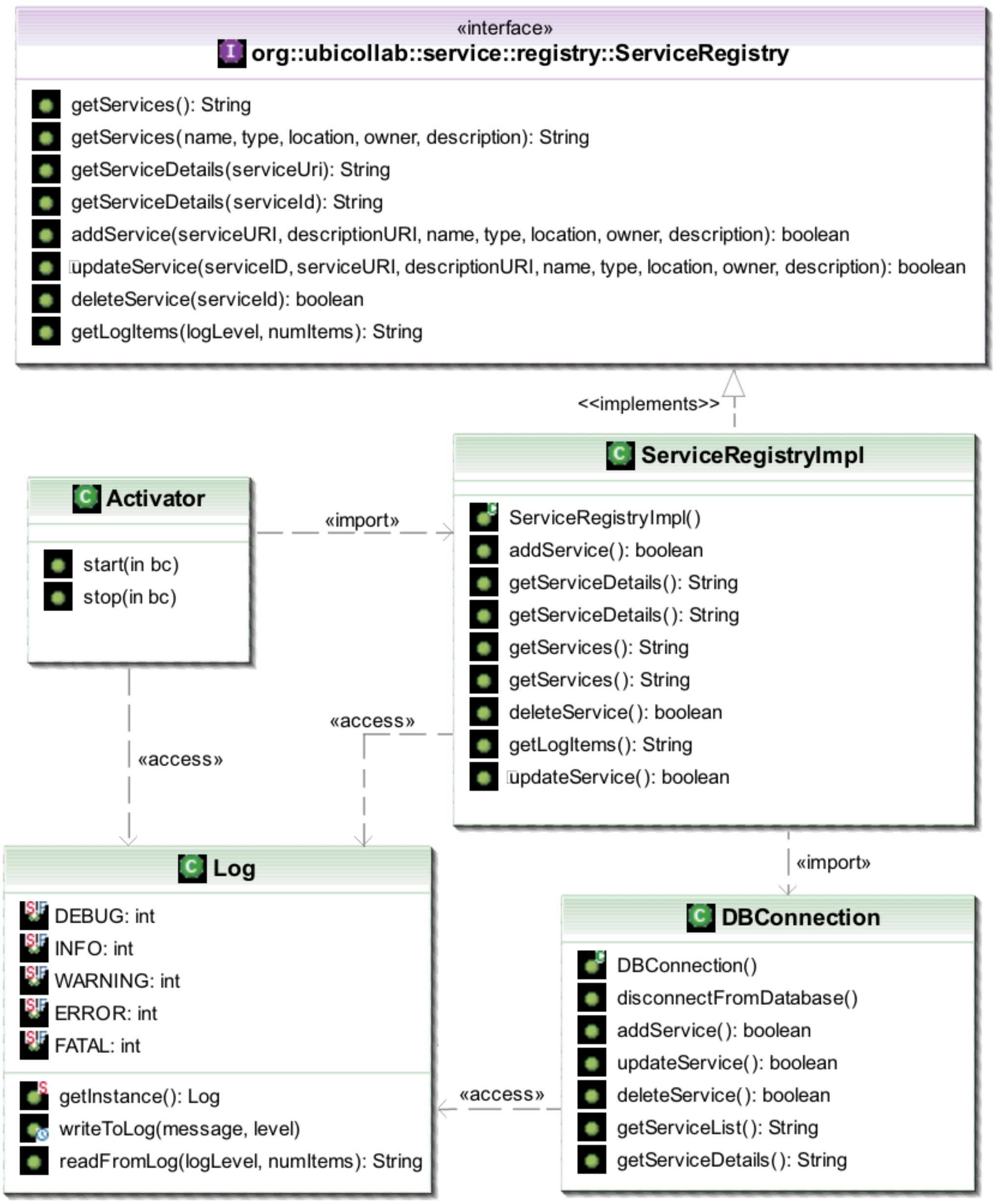
In order to provide Bluetooth connectivity, the RFIDPlugin relies on the Franson Bluetools package. This package needs to be installed prior to using the component. The necessary DLL files can also be bundled with the RFIDPlugin component. This has been described in more details in Appendix G.

5.2.3 Service Registry

Service Registry (SR) is an UbiCollab platform component implemented in Java for the OSGi platform. It provides a centralized registry where services can be advertised and a Web-service interface (SOAP) for clients to query and manage the contents in the registry. The complete API for this Web-service interface is described in detail in Appendix C.

As can be seen from the class diagram presented in Figure 5.7, this is a quite simple component with only a few classes. The `ServiceRegistryImpl` class implements the `DiscoveryManager` interface, and it is this class which exposes its public methods in the SOAP interface. The simplicity of the component comes from the `DBConnection` class. In this implementation `DBConnection` uses a MySQL database for data storage. Much of the functionality needed to resolve queries is already provided through the SQL interface provided by the database.

Figure 5.7: Class diagram for Service Registry


«interface»
org::ubicollab::service::registry::ServiceRegistry
getServices(): String
getServices(name, type, location, owner, description): String
getServiceDetails(serviceUri): String
getServiceDetails(serviceId): String
addService(serviceURI, descriptionURI, name, type, location, owner, description): boolean
updateService(serviceID, serviceURI, descriptionURI, name, type, location, owner, description): boolean
deleteService(serviceId): boolean
getLogItems(logLevel, numItems): String
<<implements>>
ServiceRegistryImpl
ServiceRegistryImpl()
addService(): boolean
getServiceDetails(): String
getServiceDetails(): String
getServices(): String
getServices(): String
deleteService(): boolean
getLogItems(): String
updateService(): boolean
Activator
start(in bc)
stop(in bc)
«import»
«access»
«access»
Log
DEBUG: int
INFO: int
WARNING: int
ERROR: int
FATAL: int
getInstance(): Log
writeToLog(message, level)
readFromLog(logLevel, numItems): String
«import»
DBConnection
DBConnection()
disconnectFromDatabase()
addService(): boolean
updateService(): boolean
deleteService(): boolean
getServiceList(): String
getServiceDetails(): String
«access»

To improve portability with regard to data source, all the specifics related to the data source is contained in one class (`DBConnection`). If a switch to a more lightweight data storage solution was considered, only the `DBConnection` class would need replacement.

The connection parameters and database script needed to set up a database that complies with what the Service Registry defaults to, is included in Appendix G.

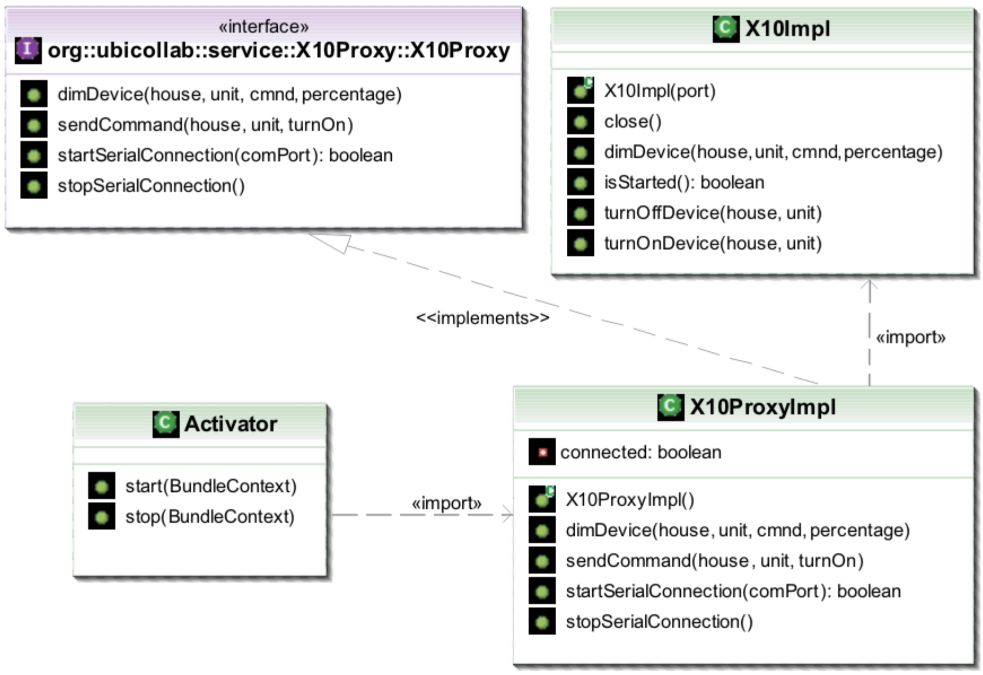
5.2.4 X10 Proxy service and demonstrator

The X10 proxy service was created to demonstrate UbiCollab's integration with the physical environment. This service interfaces with a "cm11a X10 control device" (cm11a) through the RS232 serial interface, and provide a Web-service interface where X10 devices within range of the cm11a can be controlled. In order to provide this functionality and connectivity two packages external to UbiCollab have been used. These packages are:

- For X10 connectivity the *jpeterson library*[1] was used. This is available for download as part of the "CruiseControl framework"[2] or directly from the creator's website[3]
- For serial connectivity the *Java Comm API* was used. This package is not a standardized part of Java SDK and must be installed manually. Details regarding this package and installation instructions is given in Appendix F.

The class diagram for this component is displayed in Figure 5.8. The two essential classes in this diagram are `X10ProxyImpl` which implements the `X10Proxy` interface and exposes its methods in a Web-service interface, and the X10Impl which implements the event driven system offered by the jpeterson package. The serial connection with the cm11a device is also started in the X10Impl class by the constructor. The COM port to use must be provided as a parameter for the constructor, as there is no known way to detect which COM port the cm11a device is connected to.

Figure 5.8: Class diagram for X10 Proxy service


«interface»
org::ubicollab::service::X10Proxy::X10Proxy
dimDevice(house, unit, cmnd, percentage)
sendCommand(house, unit, turnOn)
startSerialConnection(comPort): boolean
stopSerialConnection()
X10Impl
X10Impl(port)
close()
dimDevice(house, unit, cmnd, percentage)
isStarted(): boolean
turnOffDevice(house, unit)
turnOnDevice(house, unit)
<<implements>>
«import»
Activator
start(BundleContext)
stop(BundleContext)
«import»
X10ProxyImpl
connected: boolean
X10ProxyImpl()
dimDevice(house, unit, cmnd, percentage)
sendCommand(house, unit, turnOn)
startSerialConnection(comPort): boolean
stopSerialConnection()

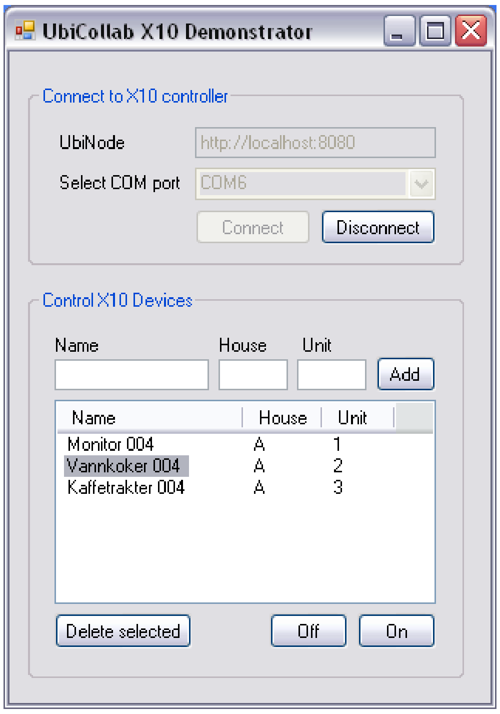
Also note the `connected` variable in the X10ProxyImpl class. This is used to maintain control of whether a serial connection is open or not. If an attempt is

[1]Pure Java implementation of the X10 protocol, created by Jesse Peterson
[2]http://cruisecontrol.sourceforge.net/ (Visisted 10.06.2007)
[3]http://www.jpeterson.com/x10/x10.zip (Visisted 10.06.2007)

made to open a serial connection which is already open, the Java Virtual Machine will freeze. This will also happen if the service tries to exit while the connection is still open. For these reasons it is important to maintain control with the connection state.

In order to demonstrate the X10Proxy a simple demonstration GUI has also been created. This GUI is displayed in Figure 5.9. As the figure shows some default connection parameters are provided, but can be changed.

Figure 5.9: Demonstrator GUI for the X10 Proxy service



Name	House	Unit
Monitor 004	A	1
Vannkoker 004	A	2
Kaffetrakter 004	A	3

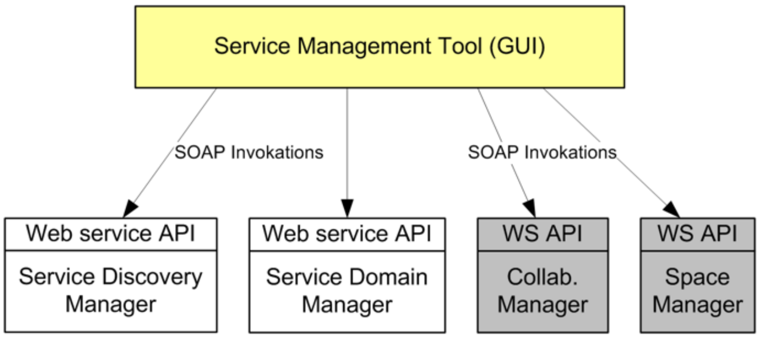
X10 devices use an addressing scheme that consist of a “house” and a “unit” code. In order to more easily refer to appliances being controlled, the GUI provides the possibility to connect the individual address (house and unit) with an easy to remember name like “Vannkoker 004”. When such a reference is created it is persistently stored in a file.

5.2.5 Service Management Tool

The Service Management Tool (SMT) have been designed and implemented with a dual purpose. It will both serve as a demonstrator GUI and as a tool in UbiCollab. It provides end-users with a user-friendly environment where they can find (discover) services, install these in their service domain, and share (publish) these services to make them available in collaboration instances where the users are members.

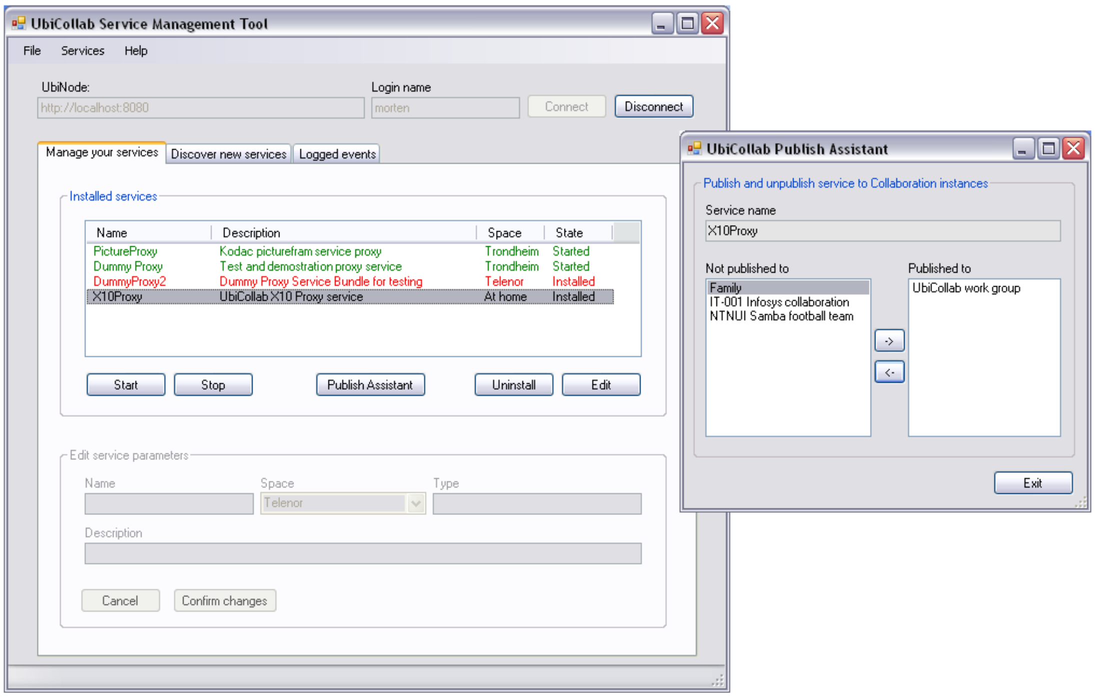
In order to achieve this functionality SMT consumes four different Web-services, as illustrated in Figure 5.10. This application is multi-threaded and runs functionality related to Discovery Manager in one thread, while functionality related to "Collaboration Manager" and "Collaboration Space" runs in its own thread. Management of services is handled in the application main thread.

Figure 5.10: Services consumed by UbiCollab Service Management Tool



To illustrate the functionality offered to the user by this application, some screen shots will be used. In Figure 5.11 the "Manage your services" tab in the SMT application is displayed. When SMT has been started the first thing the user will need to do is select an UbiNode to connect to, and enter a login name. The SMT "Manage your services" tab in the figure shows the services in the user's service domain (located on the selected UbiNode, here "localhost"), after the "Connect" button has been pressed.

Figure 5.11: Service management in UbiCollab Service Management Tool



UbiNode: http://localhost:8080
Login name: morten

Installed services

Name	Description	Space	State
PictureProxy	Kodac picturefram service proxy	Trondheim	Started
Dummy Proxy	Test and demostration proxy service	Trondheim	Started
DummyProxy2	Dummy Proxy Service Bundle for testing	Telenor	Installed
X10Proxy	UbiCollab X10 Proxy service	At home	Installed

Edit service parameters

Name:
Space: Telenor
Type:
Description:

Publish and unpublish service to Collaboration instances

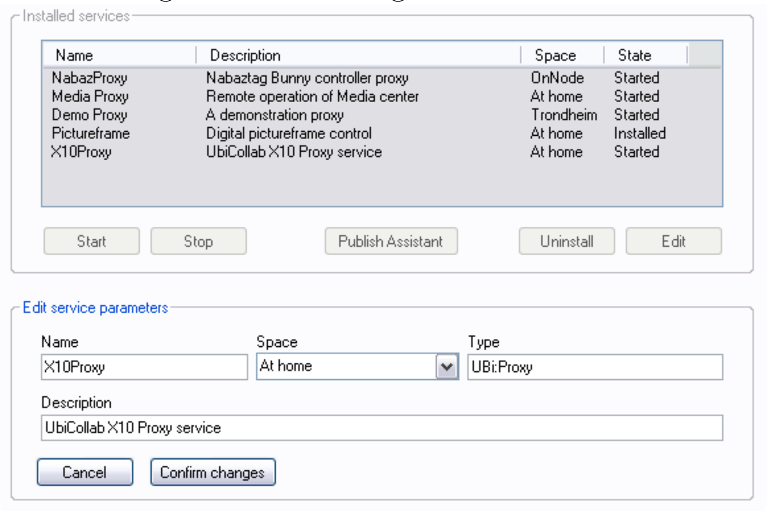
Service name: X10Proxy

Not published to	Published to
Family	UbiCollab work group
IT-001 Infosys collaboration	
NTNUI Samba football team	

In order to start or stop a service manually, it can be selected from the list of installed services and the "Start" or "Stop" button pressed. If a service proxy needs to be uninstalled this is done in a similar fashion, except that a confirm dialog will be displayed. To publish a service it must be selected before the button labeled "Publish Assistant" is pressed. This action will open the "UbiCollab Publish Assistant" (displayed in the figure) where the selected service can be published to, or unpublished from, collaboration instances (CI) in which the user is a member.

To edit the properties of a service it must be selected and the "Edit" button pressed. The "Installed services" area will then become inactive (greyed out) and the selected service will become editable in the "Edit service parameters" area. The GUI state at this point is displayed in Figure 5.12.

Figure 5.12: Editing a service in SMT


Installed services

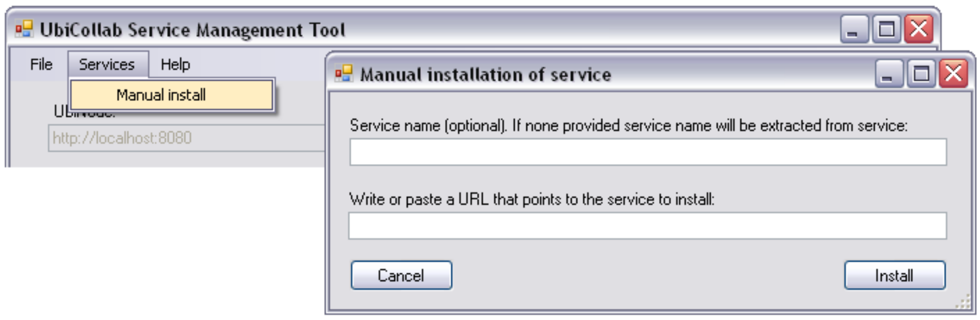
Name	Description	Space	State
NabazProxy	Nabaztag Bunny controller proxy	OnNode	Started
Media Proxy	Remote operation of Media center	At home	Started
Demo Proxy	A demonstration proxy	Trondheim	Started
Pictureframe	Digital pictureframe control	At home	Installed
X10Proxy	UbiCollab X10 Proxy service	At home	Started

Edit service parameters

Name: X10Proxy
Space: At home
Type: UBi:Proxy
Description: UbiCollab X10 Proxy service

SMT offers two ways to install service proxies. If the location of the bundle is known, a manual service install can be performed. This situation is displayed in Figure 5.13. The other way to install a service is to search for it (discovery process) and then select to install it. This process will be described in the next step.

Figure 5.13: Manual installation of service


UbiCollab Service Management Tool

http://localhost:8080

Manual installation of service

Service name (optional). If none provided service name will be extracted from service:

Write or paste a URL that points to the service to install:

To discover services the tabular "Discover new services" must be selected in SMT. This tabular is displayed in the cropped screenshot in Figure 5.14. The "Discover new services" tabular can be used to register searches (client requests) for services with the Service Discovery Manager, and discovered services will be displayed here. In the screenshot three services has been discovered using the RFID IDBlue pen. The service named "X10Proxy" has been double clicked and a dialogue named "Service Information" is opened. It shows the details for this service and offers the possibility to install it (because it is of type Ubi:Proxy). In the screenshot the "Install" button has been pressed, the service was installed and a message box prompting this to the user is displayed.

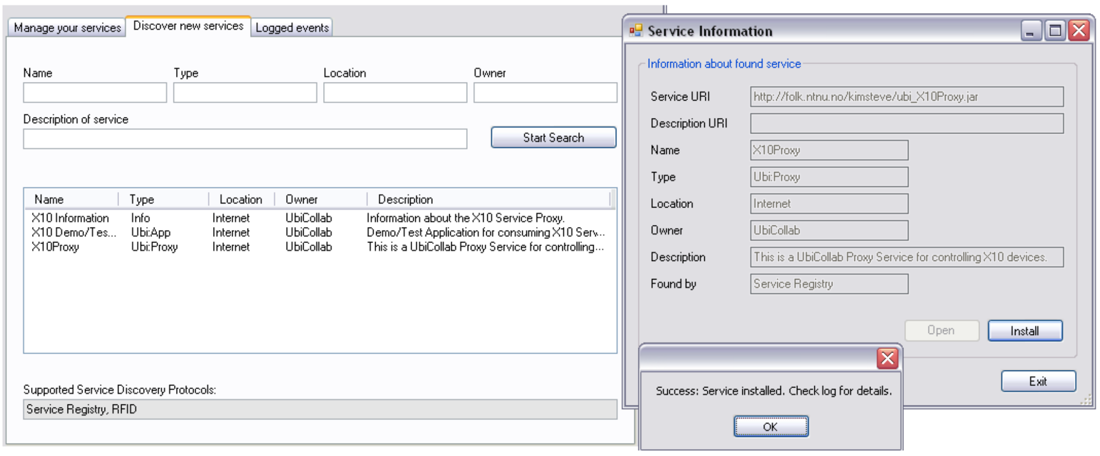


Figure 5.14: Service discovery in UbiCollab Service Management Tool

Name	Type	Location	Owner	Description
X10 Information	Info	Internet	UbiCollab	Information about the X10 Service Proxy.
X10 Demo/Tes...	Ubi:App	Internet	UbiCollab	Demo/Test Application for consuming X10 Serv...
X10Proxy	Ubi:Proxy	Internet	UbiCollab	This is a UbiCollab Proxy Service for controlling...

Supported Service Discovery Protocols: Service Registry, RFID

Service Information

Information about found service

Service URI	http://folk.ntnu.no/kimsteve/ubi_X10Proxy.jar
Description URI	
Name	X10Proxy
Type	Ubi:Proxy
Location	Internet
Owner	UbiCollab
Description	This is a UbiCollab Proxy Service for controlling X10 devices.
Found by	Service Registry

Success: Service installed. Check log for details.

Finally, SMT provides users with the possibility to view the logs from Service Domain Manager and Service Discovery Manager. This was deemed an important feature which would enable users to feel in control of what are happening in their service domain. The "Logged events" tabular is displayed in Figure 5.15. The users are able to select the log level of interest (Debug, Info, Warning and Fatal), the number of logged items to display and which log to view. In the screenshot the debug log from Service Domain Manager is displayed.

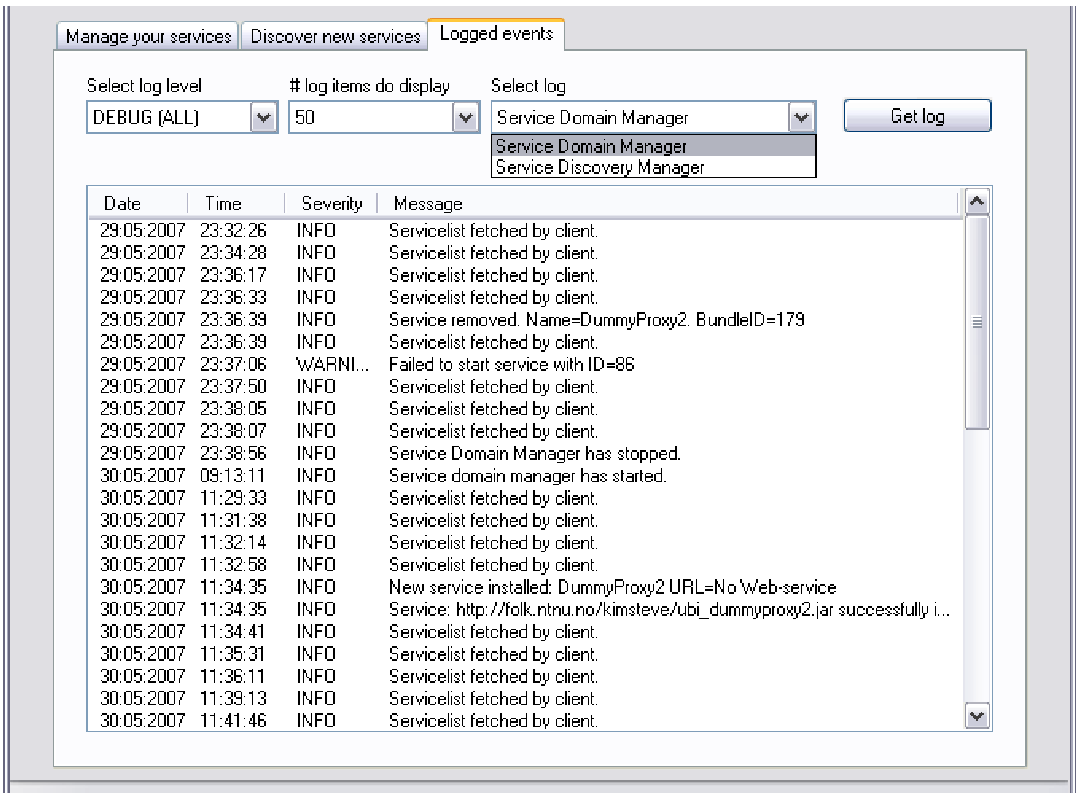
5.3 Requirements not implemented

During the implementation phase, some of the functional requirements were not implemented for various reasons. In this section these requirements will be pointed out. The reason why they were not implemented, and suggestions as to how this should be handled is given. The requirements that not have been implemented are:

FR-S4 "A user must not be allowed to discover a service which is not visible to him or her" (M,H).
Since there is no solution for a visibility or security layer in UbiCollab at the time this project was carried out, this requirement was omitted. Applications like UbiBuddy[1] correctly display services which should be visible

[1]UbiCollab demonstrator application

Figure 5.15: Displaying logged events in SMT



Select log level: DEBUG (ALL)
# log items do display: 50
Select log: Service Domain Manager

```
Date        Time      Severity  Message
29:05:2007  23:32:26  INFO      Servicelist fetched by client.
29:05:2007  23:34:28  INFO      Servicelist fetched by client.
29:05:2007  23:36:17  INFO      Servicelist fetched by client.
29:05:2007  23:36:33  INFO      Servicelist fetched by client.
29:05:2007  23:36:39  INFO      Service removed. Name=DummyProxy2. BundleID=179
29:05:2007  23:36:39  INFO      Servicelist fetched by client.
29:05:2007  23:37:06  WARNI...  Failed to start service with ID=86
29:05:2007  23:37:50  INFO      Servicelist fetched by client.
29:05:2007  23:38:05  INFO      Servicelist fetched by client.
29:05:2007  23:38:07  INFO      Servicelist fetched by client.
29:05:2007  23:38:56  INFO      Service Domain Manager has stopped.
30:05:2007  09:13:11  INFO      Service domain manager has started.
30:05:2007  11:29:33  INFO      Servicelist fetched by client.
30:05:2007  11:31:38  INFO      Servicelist fetched by client.
30:05:2007  11:32:14  INFO      Servicelist fetched by client.
30:05:2007  11:32:58  INFO      Servicelist fetched by client.
30:05:2007  11:34:35  INFO      New service installed: DummyProxy2 URL=No Web-service
30:05:2007  11:34:35  INFO      Service: http://folk.ntnu.no/kimsteve/ubi_dummyproxy2.jar successfully i...
30:05:2007  11:34:41  INFO      Servicelist fetched by client.
30:05:2007  11:35:31  INFO      Servicelist fetched by client.
30:05:2007  11:36:11  INFO      Servicelist fetched by client.
30:05:2007  11:39:13  INFO      Servicelist fetched by client.
30:05:2007  11:41:46  INFO      Servicelist fetched by client.
```

to a given user, but currently there are no security mechanisms to enforce this behavior.

FR-S9 "The solution should log events related to service discovery operations" (M,M).
All the basic/core components perform logging of these operations, and these logs are accessible to the end-users. The exception is the Service Registry Tool, which does not provide the functionality to view the Service Registry Log. This functionality should be added to this tool.

FR-O4 "The solution must provide access to service information like state, visibility and user access" (M,H).
As explained earlier, a viability layer is not yet implemented in UbiCollab. In addition, there is no possibility to control and log user-access to (proxy) services with the current solution. A solution where every service itself is responsible for access control and logging can not be seen as a flexible and viable solution. Functionality allowing end-users to set and view access information was deemed important in the autumn report. It allows end-users to feel in control of their service domain. A project aiming to create a viable solution for this has been proposed in Section 7.3.

FR-O5 "The service must be automatically unpublished when the service is uninstalled" (M,M)

This requirement was assigned to the Service Management Tool, and was not implemented because of time constraints. The implication of this is that users will have to unpublish services manually before they are uninstalled, or there will be inconsistency in the collaboration instance database. To resolve this, the Service Domain Manager (SDM) should check if the service is published when the uninstall method is invoked. If published, the SDM should not allow the service to be uninstalled, write a log message, and return a message to the client. In addition the functionality of the Service Domain Manager should be extended to handle situations where a service (bundle) has been unpublished either by other tools or manually (outside of the UbiCollab domain).

Chapter 6

Evaluation

"*Optimism is an occupational hazard of programming, feedback is the treatment.*"
- Kent Beck

In this chapter the work done in this project is evaluated. Both the implemented solution and the research value of the work are evaluated. An evaluation of the implemented components against the requirement specification is not deemed necessary, as divergences were explained in Section 5.3.

The chapter describes how the implemented platform components have been evaluated, and how the applications created for evaluation purposes are used for this. As part of the evaluation, applications created by other groups working on different parts of the UbiCollab platform uses components from this work, and visa versa.

Moreover, the research value of the project will also be evaluated and results pointed out. This will also be related to the work done in the autumn project and the autumn report. Finally some suggestions as to how the implemented solution can be improved are given.

In Section 6.1 an overview of how the different platform components have been evaluated are given, before the different parts of the evaluation are described in more detail.

Then, in Section 6.2 an evaluation of whether or not this work has lead to any advancements in related state-of-the-art. Here, it is suggested that three different advancements have been made as a result of this work.

Finally, in Section 6.3 some suggestions for improvement of the evaluated solution are presented.

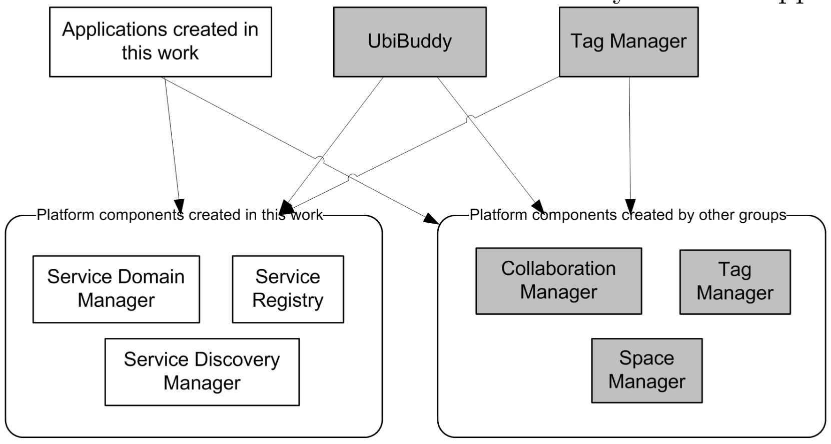
6.1 Evaluation overview

As described in Section 2.6, the biggest problem associated with evaluating platform components is that interaction with these happen through the use of intermediate applications [3]. For this reason a set of demonstration and test applications was proposed and has been implemented.

Another element that should be evaluated is how well the newly created components work with other platform components. Evaluation of this is a key issue to ensure full compatibility among the platform components. In order to provide this form of evaluation the platform components created have been used by applications created by other groups working on UbiCollab. Also, the Service Management Tool utilizes components created by other groups working on UbiCollab.

In addition to being used by the test and demonstration applications proposed and implemented in this work, the APIs provided by the components implemented have been used by two other applications created as part of other groups master thesises. This is displayed in Figure 6.1.

Figure 6.1: An overview of how this work is used by different applications


Applications created in this work
UbiBuddy
Tag Manager
Platform components created in this work
Service Domain Manager
Service Registry
Service Discovery Manager
Platform components created by other groups
Collaboration Manager
Tag Manager
Space Manager

The figure gives an overview of how the applications implemented in this project uses work created by other groups, and how applications created by other groups uses the APIs provided by this work's platform components. Work done by other groups is displayed in gray.

In the following sections, the three different evaluations which have been performed are described in more detail:

In Section 6.1.1 a demonstration of the implemented solution is described. In the test both platform components and demonstration applications created in this project was used. This demonstration was held for the people which are part of the UbiCollab and ASTRA teams at NTNU.

Then, in Section 6.1.2 the functionality of the UbiBuddy application is described. This is a demonstration application for the UbiCollab platform which is able to display awareness information in collaboration instances. This application uses the Domain Manager and Service Management Tool from this work.

Next, in Section 6.1.3 a tag management system which provide the functionality to describe services using short comments (tags) is described. This system uses the service discovery subsystem provided in this solution to discover services.

6.1.1 Demonstration

The implemented solution was demonstrated for both the ASTRA team and the UbiCollab team at a Workshop the 30th of Mai 2007 at IDI/NTNU. As part of this demonstration a presentation was held to give an overview of the solution, before the different components were demonstrated separately. Finally, a demonstration scenario involving most parts of the solution was walked through. In the reminder of this section the demonstrated scenario will be presented, before the steps taken during the presentation is described and screenshots displayed.

6.1.1.1 Demonstration scenario

Peter has been using UbiCollab for about a month now, and he has recently made a pact with his friend Tore which also uses UbiCollab. Peter met Tore through the local springer-spaniel club, and they both love dogs. They have agreed to both make a lamp in their living room available for each other, to signal when they are interested in taking the dogs for a walk.

Peter has just returned home from the post office, having picked up the X10 home automation system he ordered last week. He opens the package and notices a tag on the control device. He then starts up his Service Management Tool and uses his physical discovery device to read the tag on the X10 control device. Immediately three services pop up in his Management Tool.

Peter sees that one of the services is an information service and starts it. On the information page he finds information about how to connect the X10, along with feedback and lessons learned from other users and developers. He starts the installation, following the instructions on how to connect it. Once connected, he installs the X10 proxy he discovered a minute ago. To make sure the installation turned out OK, he selects the possibility to view the log in the Management Tool, and it looks fine.

The third service discovered was a small test application to ensure the X10 device and service proxy is functional. He installs this too, and tests that he is able to turn the lamp on and off from the application. It works fine. Finally, Peter selects the X10Proxy service and start the publishing assistant in his Service Management Tool. He then select the "Dogwalker's association" collaboration instance they created earlier, and moves it from "Not published" to "Published", and his lamp then becomes available to Tore.

6.1.1.2 Scenario Walktrough

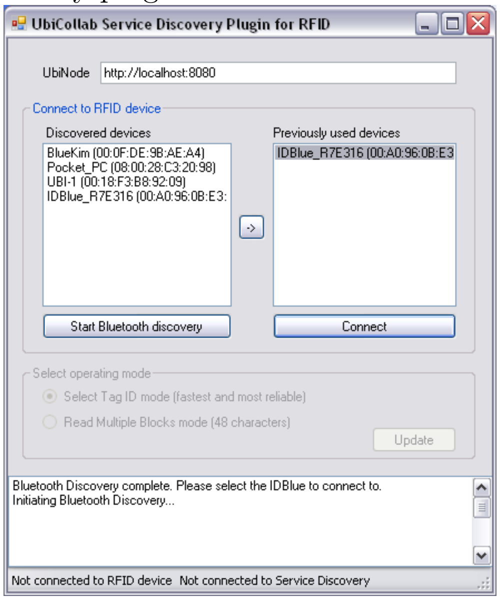
There were some prerequisites that had to be in place in order to play out this scenario. The prerequisite steps needed are explained here:

- An RFID tag had to be encoded with a URI pointing to the XML-file with the description of the three service advertisements.
- The XML-file containing the three service advertisements had to be made.
- A collaboration instance named "Dogwalker's association" had to be made.
- The X10 demo application had to be installed on the demo computer, as the auto-install feature for applications has not yet been implemented in the Service Manage Tool.

In a visionary "real world UbiCollab scenario" the two first steps would already be in place by the manufacturer. The name "Peter" from the scenario is used in this walkthrough, when end-user actions are performed.

The scenario walkthrough starts off by Peter opening the Service Management Tool, and then the RFID plugin. He initiates the Bluetooth discovery, and when finished selects the correct IDBlue RFID pen to connect to. A screenshot of this is displayed in Figure 6.2.

Figure 6.2: The discovery plugin for RFID has found the correct RFID pen


UbiCollab Service Discovery Plugin for RFID

UbiNode: http://localhost:8080

Connect to RFID device

Discovered devices:
BlueKim (00:0F:DE:9B:AE:A4)
Pocket_PC (08:00:28:C3:20:98)
UBI-1 (00:18:F3:B8:92:09)
IDBlue_R7E316 (00:A0:96:0B:E3:

Previously used devices:
IDBlue_R7E316 (00:A0:96:0B:E3

Select operating mode
(● Select Tag ID mode (fastest and most reliable)) ○ Read Multiple Blocks mode (48 characters)

```
Bluetooth Discovery complete. Please select the IDBlue to connect to.
Initiating Bluetooth Discovery...
```

Not connected to RFID device Not connected to Service Discovery

Then, the connect button in the RFID plugin is pressed, and the plugin registers with the Discovery Manager. The Service Management Tool detects this and "RFID" is added to the list of supported service discovery protocols. Next, the IDBlue pen is used to read the RFID tag, and the URI discovered on the tag is being resolved. The XML-file with three service advertisements is discovered, and the three service advertisements are added to the service list for Peter's Service Management Tool.

The Service Management Tool fetches the advertisements from Discovery Manager and displays them in the GUI. Peter double-clicks the information service and a Web-page opens. Peter reads the information there and connects the X10 devices according to the instructions. Then he double-clicks the X10 Proxy and the service advertisement is displayed, with an option to install the service. Peter presses the "Install" button and a message pops up telling him the proxy was successfully installed. The state of the GUI at this point is displayed in Figure 6.3.

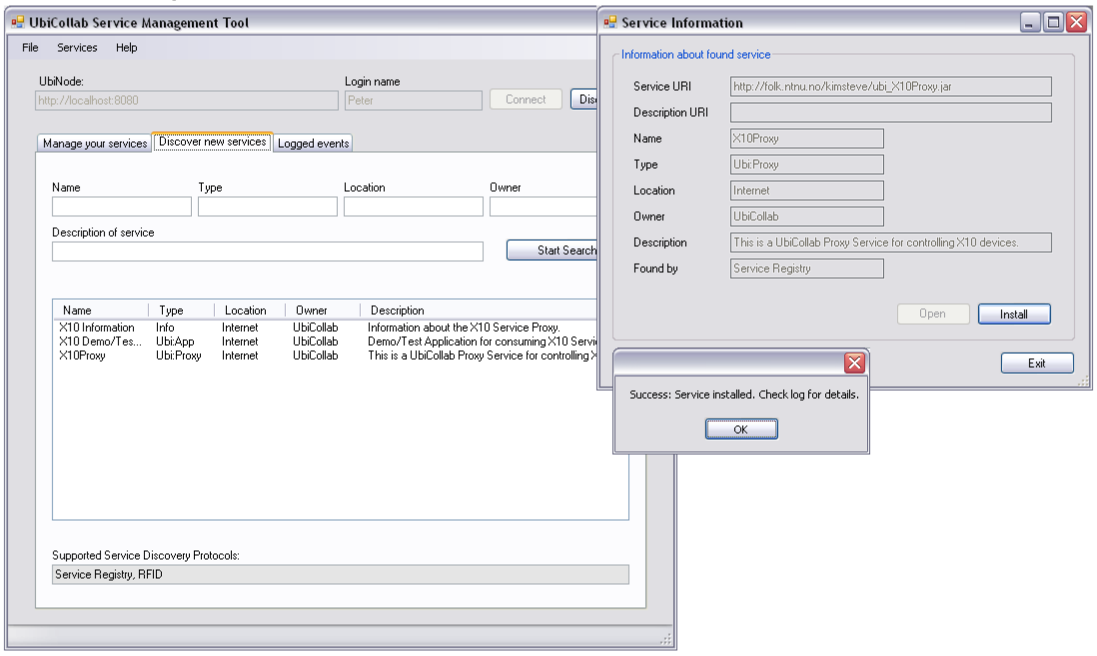


Figure 6.3: The service advertisements have been discovered

UbiNode: http://localhost:8080
Login name: Peter

Name | Type | Location | Owner
Description of service

Name	Type	Location	Owner	Description
X10 Information	Info	Internet	UbiCollab	Information about the X10 Service Proxy.
X10 Demo/Tes...	Ubi:App	Internet	UbiCollab	Demo/Test Application for consuming X10 Servi
X10Proxy	Ubi:Proxy	Internet	UbiCollab	This is a UbiCollab Proxy Service for controlling X

Supported Service Discovery Protocols: Service Registry, RFID

Information about found service

Service URI	http://folk.ntnu.no/kimsteve/ubi_X10Proxy.jar
Description URI	
Name	X10Proxy
Type	Ubi:Proxy
Location	Internet
Owner	UbiCollab
Description	This is a UbiCollab Proxy Service for controlling X10 devices.
Found by	Service Registry

Success: Service installed. Check log for details.

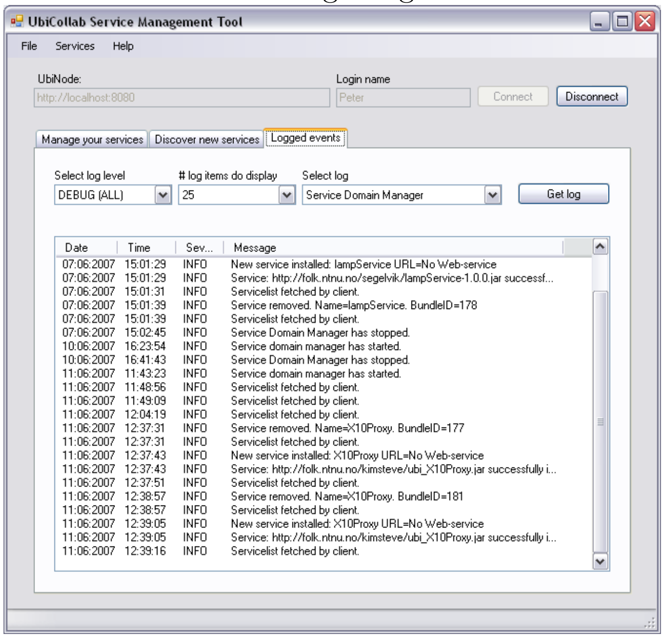
Peter has the possibility to see the details of the installation process by clicking on the log botton. The GUI state for the "Logged events" tabular at this point is displayed in Figure 6.4.

Next, Peter starts up and connects the X10 test application to the X10Proxy and adds the X10 devices he has installed in his house. He selects the living room lamp and turns it on and off a couple of times to ensure it works. The state of the X10 demonstrator application at this time is displayed in Figure 6.5.

Now, Peter changes the tab in his Service Management Tool to "Manage your services" and selects the X10Proxy service, which is now part of the services he has installed on his UbiNode. Peter presses the "Publish Assistant" button and a new dialog is opened with the possibility to publish the selected service to the collaboration instances he is a member of. Here, the "Dogwalker's association" is selected and moved to the "Published to" section. The dialog at this stage is shown in Figure 6.6.

The X10 controlled lamp have now been made available for Tore, as he is a member of the "Dogwalker's association" collaboration instance. From this point

Figure 6.4: The Domain Manager log after a service is installed

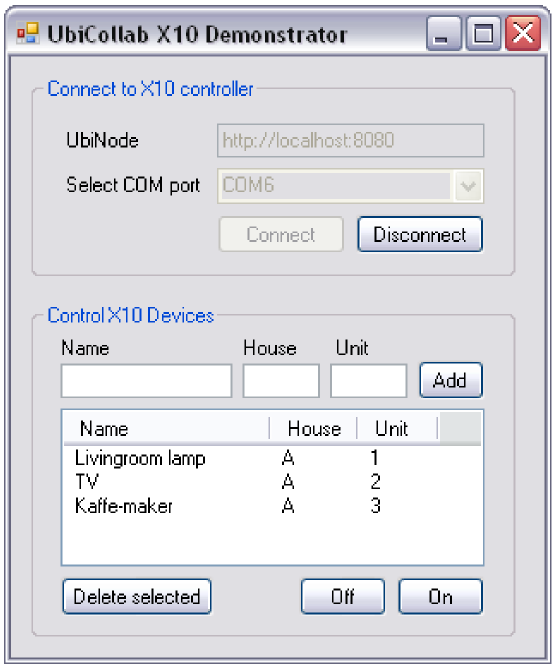


UbiNode: http://localhost:8080
Login name: Peter

Select log level: DEBUG (ALL)
# log items do display: 25
Select log: Service Domain Manager

```
Date        Time      Sev...  Message
07:06:2007  15:01:29  INFO    New service installed: lampService URL=No Web-service
07:06:2007  15:01:29  INFO    Service: http://folk.ntnu.no/segelvik/lampService-1.0.0.jar successf...
07:06:2007  15:01:31  INFO    Servicelist fetched by client.
07:06:2007  15:01:39  INFO    Service removed. Name=lampService. BundleID=178
07:06:2007  15:01:39  INFO    Servicelist fetched by client.
07:06:2007  15:02:45  INFO    Service Domain Manager has stopped.
10:06:2007  16:23:54  INFO    Service domain manager has started.
10:06:2007  16:41:43  INFO    Service Domain Manager has stopped.
11:06:2007  11:43:23  INFO    Service domain manager has started.
11:06:2007  11:48:56  INFO    Servicelist fetched by client.
11:06:2007  11:49:09  INFO    Servicelist fetched by client.
11:06:2007  12:04:19  INFO    Servicelist fetched by client.
11:06:2007  12:37:31  INFO    Service removed. Name=X10Proxy. BundleID=177
11:06:2007  12:37:31  INFO    Servicelist fetched by client.
11:06:2007  12:37:43  INFO    New service installed: X10Proxy URL=No Web-service
11:06:2007  12:37:43  INFO    Service: http://folk.ntnu.no/kimsteve/ubi_X10Proxy.jar successfully i...
11:06:2007  12:37:51  INFO    Servicelist fetched by client.
11:06:2007  12:38:57  INFO    Service removed. Name=X10Proxy. BundleID=181
11:06:2007  12:38:57  INFO    Servicelist fetched by client.
11:06:2007  12:39:05  INFO    New service installed: X10Proxy URL=No Web-service
11:06:2007  12:39:05  INFO    Service: http://folk.ntnu.no/kimsteve/ubi_X10Proxy.jar successfully i...
11:06:2007  12:39:16  INFO    Servicelist fetched by client.
```

Figure 6.5: The X10 test application



UbiNode: http://localhost:8080
Select COM port: COM6

Name	House	Unit
Livingroom lamp	A	1
TV	A	2
Kaffe-maker	A	3

on, Tore can signal his willingness to take a walk by turning the lamp in Peter's living room on and off, in this way generating an ambient signal to Peter.

6.1.1.3 Feedback

During this demonstration all the demonstrated components worked as can be expected, without any problems. The feedback received after the demonstration

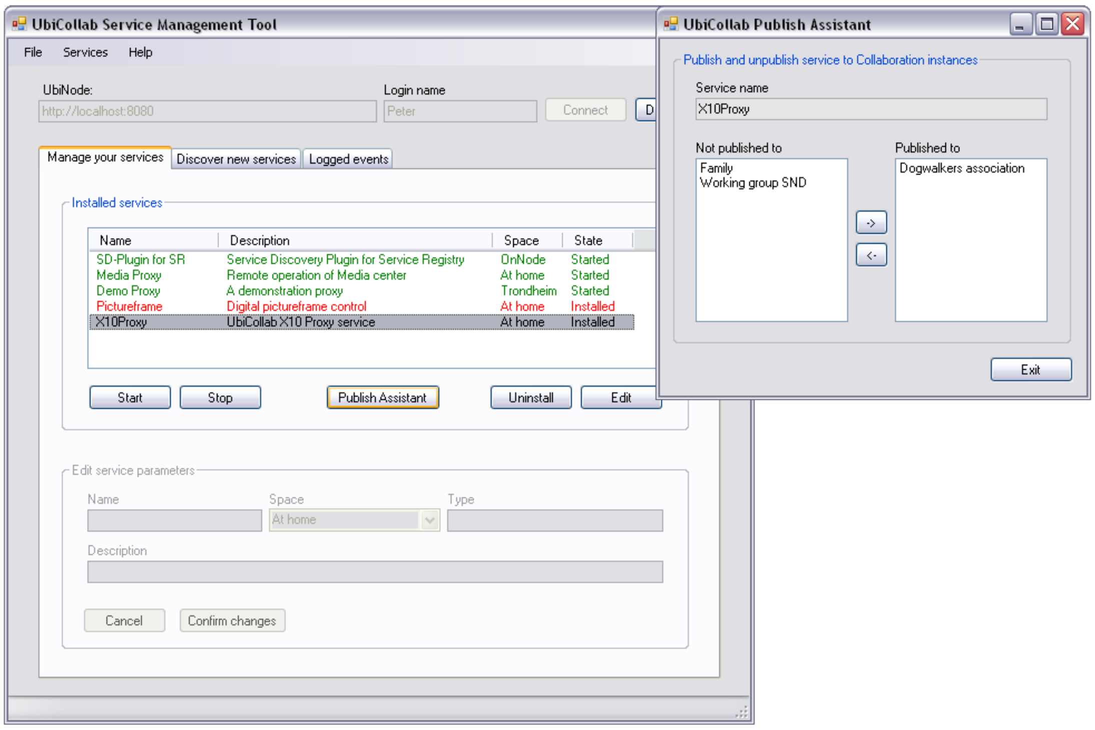


Figure 6.6: The X0Proxy has been published to the “Dogwalkers association”

UbiCollab Service Management Tool

UbiNode: http://localhost:8080

Login name: Peter

Installed services

Name	Description	Space	State
SD-Plugin for SR	Service Discovery Plugin for Service Registry	OnNode	Started
Media Proxy	Remote operation of Media center	At home	Started
Demo Proxy	A demonstration proxy	Trondheim	Started
Pictureframe	Digital pictureframe control	At home	Installed
X10Proxy	UbiCollab X10 Proxy service	At home	Installed

Edit service parameters

Space: At home

UbiCollab Publish Assistant

Publish and unpublish service to Collaboration instances

Service name: X10Proxy

Not published to	Published to
Family	Dogwalkers association
Working group SND	

suggested that the solution was perceived as being very user-friendly. Also how well the different components worked together were pointed out.

6.1.2 UbiBuddy

UbiBuddy is a demonstration application designed to show some of the features provided by UbiCollab. The goal of UbiBuddy is to provide a simple buddy list application[1] which is able to show presence information about people. More information about this component is available in the UbiCollab Whitepaper [5]. The application has been developed in parallel with this work, as part of another master thesis [19].

The application is integrated with the work done in this project in the following ways:

- UbiBuddy integrates a button which opens the Service Management Tool, to allow services to be published. This also gives the user access to the service discovery subsystem through UbiBuddy, and enables them to discover and install new services.
- One of the key aspects of UbiBuddy is the ability to view services published by your buddies. These services must be published to collaboration

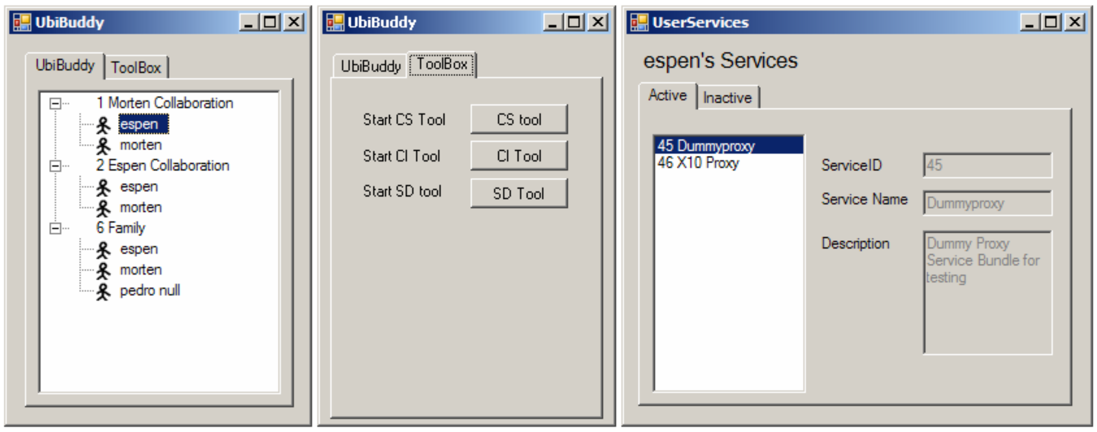
[1]Well known commercial buddy list applications include AOL Instant Messenger, MSN Messenger etc.

instances you both are members of to be visible.

- In order to show real time status of services (active, inactive), UbiBuddy connects to Domain Manager for this information.

Three screenshots showing the essential functionality are UbiBuddy is displayed in Figure 6.7. To the left the main window in UbiBuddy is displayed, with three collaboration instances (CI) starting with "Morten Collaboration". As elements under each CI the members of that CI are displayed.

Figure 6.7: UbiBuddy application main window and service window



To the right in the figure, a screenshot of the dialog that opens when a member is double-clicked is displayed. This dialog shows the services published by the double-clicked user. Two tabs are offered, one showing the active services and one showing the inactive. When a service is selected, information about this service is displayed. Both the state information and the other service information is fetched from the Domain Manager.

The "ToolBox" tabular in the UbiBuddy main window (shown in the middle of the figure) contains a set of buttons which can be used to open related UbiTools. One of the tools available here is the Service Management Tool, in order to provide the functionality needed to manage and publish services. This can be started by pressing the "SD Tool button".

The use of Domain Manager and Service Management Tool as part of the UbiBuddy solution seems to work well. Unfortunately, due to time constraints no proper formal demonstration showing the full functionality has been held. The reason for this was that both the Service Management Tool and the publishing solution was finished at a late stage, and hence the final integration was done after the Workshop described in Section 6.1.1. However, internal testing and informal demonstrations suggest that the integrated solution works as can be expected.

6.1.3 Tag Manager

A system for tag management has also been developed in parallel with this work, as part of another master thesis [15]. The purpose of the tag management system is to enable end-users to tag services (in UbiCollab service proxies) with short descriptive comments. This system can then be used to "navigate" the tags to discover similar services or find other services tagged by a given person. As part of this work an application named "Tag Manager" has been developed.

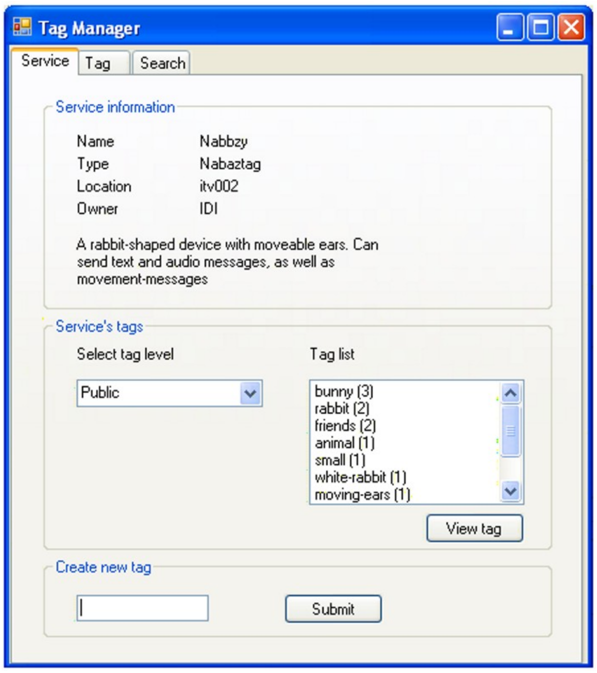
In order to discover services the "'Tag Manager" application relies on the Service Discovery Subsystem. It connects to Discovery Manager as a client, and uses the RFID Plugin and IDBlue pen to discover services. The Tag Manager uses the RFID Plugin in the "Select Tag ID mode", as opposed to what was used in the demonstration described in Section 6.1.1.

The implication of this is that only the unique identifier from the RFID tag is registered with the Discovery Manager, and this is what the client application (Tag Manager) gets. In order to resolve this identifier, the Tag Manager must register a client request containing the identifier with the Discovery Manager. The "Service Discovery Plugin for Service Registry" will then resolve this client request against the Service Registry and return the matching service advertisements (can be 0..n).

The advantage of using this operating mode with the RFID plugin, is that it is fast and easy to introduce new service advertisements. There is no need to program the tags themselves as they all have a unique ID. By using the "Service Registry Tool" it is relatively easy to add or edit service advertisements matching the tag ID.

A screenshot showing when the Tag Manager has discovered a service named "Nabbzy" is displayed in Figure 6.8. As the picture beside the application illustrates, this is a proxy service used to control a Nabaztag rabbit.

Figure 6.8: Tag Manager application in use



Service information

Name	Nabbzy
Type	Nabaztag
Location	itv002
Owner	IDI

A rabbit-shaped device with moveable ears. Can send text and audio messages, as well as movement-messages

Service's tags

Select tag level: Public

Tag list:
bunny (3)
rabbit (2)
friends (2)
animal (1)
small (1)
white-rabbit (1)
moving-ears (1)

Create new tag

The feedback received from the use of the service discovery subsystem as part of the "Tag Manager" solution have been very positive. The master student developing the solution found it easy to work with. This solution has also been demonstrated for members of ASTRA and UbiCollab teams, and the Service Discovery Subsystem performed as can be expected also there.

6.2 Advancements to state-of-the-art

In this section the value this work has added to state-of-the-art research will be described. It is believed that this work has provided three different elements which can be of value for related research fields. In the following, these three elements will be described.

First, in Chapter 3 a survey of possible technologies that would be suitable for physical discovery were performed. Three candidate technologies were selected for further analysis and they were also compared. From these three, RFID was deemed the most suitable technology, and support for this technology was implemented in the solution. Experiences from implementation and evaluation suggest this was a good choice and that it can be recommended for similar use.

This work has also presented a solution for service discovery which is able to combine traditional service discovery protocols with physical discovery. In the state-of-the-art (Chapter 5) in the autumn report a lot of related research was surveyed, without encountering any solution to this. Therefore it is believed that this work has presented a possible solution for how physical and "traditional" discovery can be integrated. Evaluation suggests that also this solution works very well.

In addition, the user-friendly service discovery and management solution implemented in this project has strengthened the answers to the research questions proposed in the autumn report. In the autumn report it was asked what is implied by user-centered and community oriented service discovery. Based on this an analysis was performed, requirements were elicited and a high-level architecture was created. In this work that architecture has been refined in the proposed solution, which was then implemented.

The evaluation of this implementation suggests the chosen solution is both user-friendly and works well in a community oriented environment. The real power which makes the solution user-friendly is not the individual components, but the way they work together in order to provide advanced functionality and easy to use user-interfaces.

6.3 Improvement suggestions

This section will provide some suggestions for improvement of the evaluated solution. These suggestions are based mainly on the evaluation described here in this chapter, but also on informal testing and tests performed during the implementation. Suggestions expected to bring about enough work/research for a master thesis are described in future research (Section 7.3).

The first suggestion is that a more user friendly method should be used to connect to the IDBlue RFID pen. In the current solution the user need to select the pen and click the “Connect button”. A possible solution could be to add a thread to the RFID plugin that will perform a Bluetooth discovery search at given time intervals. When a pen matching the stored address is found it is automatically connected to. The plugin could also be remotely controlled by an intermediary located on the OSGi framework, and only be visible as an icon in the Windows tray icon.

A version of the plugin should also be compiled using the “.NET Compact Framework” in order to provide support for Pocket PCs and smartphones. Some simplification to the GUI would be needed first, as the compact framework do not support all GUI components used on the Windows .NET version. Details regarding the Bluetooth library needed for compact framework are described in Appendix G.

More service discovery plugins should be developed to add to the utility of the service discovery solution. Particularly the WS-Discovery protocol[1] looks interesting as many believe this protocol will take the place of UPnP and become the service discovery protocol of choice for hardware devices. The protocol is included in the new Windows Vista operating system, and practical applications

[1]http://schemas.xmlsoap.org/ws/2005/04/discovery/ (Visited 16.06.2007)

of this protocol are starting to appear[1].

Also a implementation of QR-code base plugin as described in state-of-art (Section 3.1.2) should be considered. Because of the ubiquity of camera phones this could open the door for large scale testing and demonstration of physical discovery in UbiCollab at a small price.

One problem with the Discovery Manager is the lack of an internal mechanism to detect when the same service have been discovered by more than one plugin. The result of this is that a service requester may receive two equal service advertisements, when services have been advertised several places. Such a mechanism is assumed to be fairly easy to create and should be implemented.

Another issue with the discovery manager is the way service advertisements discovered by physical discovery mechanisms are registered. Currently, all registered clients will receive these. A method should be added where clients can register their interest for physical discovery as some applications is likely to only use only networked discovery. Another option is to add a boolean parameter to the method used by clients for registration, specifying if the client is interested in physical discovery.

In addition to these suggestions, another issue which has been mentioned earlier in the report is the use of the location field in service advertisements. How this field should be used and which format should be selected is an open issue. A possible solution could be to define a format to link it with space management. The value of this field for use in a user-centered service discovery solution have been analyzed and considered to be high, and as such it should be better utilized.

[1]http://www.microsoft.com/biztalk/technologies/rfid/default.mspx (Visited 31.05.2007)

Chapter 7

Conclusions and future research

"The main purpose of science is simplicity and as we understand more things, everything is becoming simpler."
- Edward Teller

In this chapter the report will be concluded by presenting and evaluating the work that has been done. The contributions this project has yielded will be presented, and some suggestions for future research projects will be proposed.

In Section 7.1 the contributions this project has provided for UbiCollab and other related projects will be described.

Next, in Section 7.2 a short evaluation of the results from the work as well as the work process will be given.

Finally, in Section 7.3 some suggestions for future research projects within the larger UbiCollab project is presented.

7.1 Contributions

This project and the work performed has itself been a contribution to the larger UbiCollab project. The aim of the work has been to create a user-centered service discovery and management solution, suitable for use in a distributed collaborative environment. The work done has built on the autumn project which was a contribution to the ASTRA project, and many of the ideas from that project have been implemented in this work. As such, this report includes contributions to both the ASTRA project and other similar projects as well. More specifically, the individual contributions are as follows:

The first contribution is an extension of the state-of-the-art analysis performed in the autumn report. In this extension, relevant technologies that can be used to implement a user-centered physical service discovery were surveyed. The surveyed technologies were evaluated for use in UbiCollab, and RFID was deemed the most suitable technology for use as a physical discovery mechanism.

A proposal for a service management solution has also been created, implemented and evaluated. This solution has been based on ideas from the autumn report as well as requirements elicited from UbiCollab. The component created was named **Service Domain Manager**, and evaluation suggests that this component is well suited for further use in UbiCollab.

Also, a solution for a service discovery subsystem has been proposed and implemented. This subsystem has to a large extent been based on the functional requirements elicited, and the high-level architecture proposed in the autumn report. The components created to form this subsystem includes **Service Discovery Manager**, **Service Discovery Plugin for RFID**, **Service Discovery Plugin for Service Registry**, **Service Registry** and **Service Registry Tool**. Service Registry has been included as part of the service discovery subsystem, but this registry and the accompanying tool can also be seen as a subsystem of their own.

Moreover, a tool named **Service Management Tool** has been created to provide end-users the means for discovery of services, installation and management of installed services. This tool also demonstrates the platform functionality provided by the Service Domain Manager and the Service Discovery subsystem. In addition it demonstrates functionality provided by Space Manager and Collaboration Manager, which have been developed as part of another UbiCollab master thesis [19].

Finally, an X10 proxy service has been developed for test and demonstration purposes, with an accompanying demonstrator application. Although simple, this proxy service provides means to demonstrate how the UbiCollab platform integrate with the physical environment.

7.2 Evaluation

In this chapter a short evaluation of the results from the work performed, as well as the work process will be given. The goal of this work has been to design and implement a solution for service management which meets the requirements of the UbiCollab project. Much of the challenge in this project has been posed by the focus on making the solution user-centered and flexible enough to support cooperation within a variety of application domains.

After having performed a thorough evaluation of the implemented solution, the conclusion is that this work has successfully achieved its goal. A user-centered service discovery and management solution has been proposed and implemented. An evaluation of the solution has been performed through the use of the solution in other work and demonstrations, as described in Chapter 6. A more complete evaluation with end-user tests and empirical data would have been desirable, but was not feasible due to time constraints.

At the time the project assignment was agreed and a master contract was signed, it was clear that the amount of work associated with this project would be quite large. In order to finish the project in time a plan was created where the amount of time spent on the various parts was determined. As in so many other projects some time limits were exceeded and the plan needed continuous adjustment. Still, the value of following the plan has been high as it not only have provided time limits, but also served as a reminder for how much work is left.

The biggest regret is that not enough time was scheduled for integration with other groups, and development of mutual demonstration applications. The UbiBuddy application was created but if more time had been spent on this, the value of this for demonstration and evaluation purposes would have been higher. Despite this, some time has been spent on integration and all of the components created in this work have been used by other projects, as described in Chapter 6. The exception to this is the X10 demonstrator which only was used in the demonstration described in Chapter 6. However, the ASTRA team has expressed interest in using this demonstrator as part of one of their future demonstrations.

An important part of this project has been collaboration with other groups working on the platform, in addition to some collaboration with people working on the ASTRA platform. This collaboration was given a lower priority in the early phases of the project, but has been good in the late stages. The collaboration could have been better from the start if there had been regular meetings, informing each other of progress and sharing information. This issue was addressed at the Workshop held at the 30th of Mai 2007 at IDI/NTNU.

7.3 Future Research

In this chapter some ideas for future projects will be suggested. These ideas have been categorized into project suggestions aimed at extending the service discovery and management solution, and other projects. The suggested projects have come up as the result of elements that have been missed in UbiCollab during this work, and also some other interesting projects or concepts that have been discovered during the work.

Suggested research projects related to service discovery and management:

Service negotiation A valuable addition to the UbiCollab platform would be to have a generic negotiation solution for the terms a given user must agree to before he can get access to and use a service. This would allow both commercial actors and others with similar needs, to offer services in UbiCollab. Service negotiation will then be the process where the *service requester* and the *service provider* agrees on the terms for using the service. A possible solution could be that the service requester gradually requests additional parameters for the service in question. Such parameters can be quality of service, availability, use-period or cost. The service information is gradually refined sufficiently for the requester to formulate a "configuring request". Optionally, this phase may include the fulfillment of a precondition to requesting the service (e.g. registration or membership). This is what is often referred to as an "enabling condition". This work can also be linked with the use of a *service broker* as a mediator between the service provider and service requester. A service broker is usually a person or organization responsible for a registry where services are advertised. The most common registry type used for this purpose on the Internet is the UDDI-registry. In some cases the service broker can also act on behalf of the service provider and take his place. He will then handle the service negotiation with service requesters on behalf of the service provider. Many services, like Bluetooth and UPnP does not use a broker as they utilize self advertising. Such services can be represented by a *service proxy* in order to publish them in a registry and hence make them available through a service broker.

Expanding the service domain across nodes The current domain manager provides the functionality needed for management of services/bundles on one UbiNode. To enhance the service domain a solution where end-users can have several UbiNodes and manage their services across these should be considered. Dynamic deployment of services and bootstrapping of new UbiCollab devices should also be considered in the same context. One possible method for bootstrapping is to create an installer package containing both the OSGi framework and the required bundles. Another more advanced and flexible option, is to use an installer script or create an installer service/bundle. This service can then have responibilities like automatic updates etc. The Service Domain Manager already provides the basic functionality for such operations and can be extended to support this, but caution is needed to avoid tight coupling or unnecessary dependencies.

Service access control Currently, no solution is implemented to detect access to, and usage of services. Perhaps the most flexible and powerful solution could be achieved by implementing a SOAP message handler. Currently "JAX-RPC" (SOAP 1.1) is used in UbiCollab (because Axis v.1.x is being used), but the newer "JAX-WS"[1] (Java API for XML Web Services) is recommended for this use since it supports most newer WS-* protocols. Both support message handlers though, and porting from one to the other should be possible. The following benefits can be achieved by the use of SOAP message handlers:

- To do encryption and decryption on the client and service side for exchanging messages securely
- To provide access-control to Web services

[1]https://jax-ws.dev.java.net/ (Visited 13.06.2007)

- To provide auditing, logging and metric collection

A solution to this should perhaps be investigated in relation with a possible selection of a new SOAP engine, as described in the next project suggestion.

Suggested research projects not related to this work:

SOAP Provider Axis v.1.x (Axis v1) is currently used as the core engine to provide Web services in UbiCollab, but this product has been discontinued since April 2006[1]. One of the major problems with the Axis v1 is that the "rpc/encoded" format is used in SOAP encoding instead of the "document/literal" format. "rpc/encoded" has been deprecated by the WS-I Basic Profile[2], which is the baseline for inter-operable Web services. Axis v1 is for instance not inter-operable with Axis v2 (a redesign of Axis v1, with the support for SOAP1.2/REST/and other newer standards). In addition Axis v.1.2 and older versions (like the port to OSGi which is used in UbiCollab) have problems with the changes in the XML-handling that were introduced with Java v1.5.x (JAX-RPC 1.1 specifications). Since no newer OSGi port exists of Axis v1, a patched port must be used or the full Axis libraries and its dependencies must be included in the bundles created. The general impression is that continuing using Axis v1 will lead to a minefield of compatibility issues. For these reasons an **Engineering Project** with the mission to find a new SOAP provider should be initiated. If no better SOAP implementation can be found for OSGi, perhaps moving to another container should be considered?

Node discovery Some means of node discovery is sorely missed in UbiCollab. The result of this is that a combination storing addresses in text-files and users entering the address manually in text-fields in the user interface is being used by the current projects. A possible candidate that has been mentioned and that will provide the mean to communicate and collaborate in a P2P manner is JXTA[3]. The use of WS-Discovery for this purpose should also be considered, as this discovery protocol has many desirable features. An implementation of WS-Discovery in combination with a UDDI registry could prove to be an innovative and powerful solution. When a node discovery solution has been found, some of the current platform components are likely to need modifications to comply with the new solution. This part should perhaps also be included in the same project, to ensure consistency.

Security - Identity management During the work some technologies that should be considered for security and identity management have been encountered.

[1]http://ws.apache.org/axis/ (Visited 03.05.2007)

[2]http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicprofile (Visited 03.05.2007)

[3]http://www.jxta.org/ (Visited 03.05.2007)

- One that looks promising is the “Windows CardSpace”, which is an identity management system. It is built on top of the Web Services Protocol Stack (WS-*). This implies that any technology or platform which supports WS-* protocols, such as UbiCollab, can be integrated with CardSpace. Windows CardSpace offers the possibility to choose the identity being used for a particular transaction. CardSpace is part of, and seem to rely on the .NET 3.0 framework. Currently there are plugins for Internet Explorer, Netscape and Opera which support it. Even if the platform selection differs from UbiCollab, many of the ideas and concepts seem interesting.
- Another project which could be interesting is the “Higgins Trust Framework”[1]. This is an open source project within the Eclipse Foundation. The purpose of Higgins is to provide “an extensible, platform-independent, identity protocol-independent, software framework to support existing and new applications that give users more convenience, privacy and control over their identity information”.

[1]http://www.eclipse.org/higgins/ (Visited 05.06.2007)

Chapter 8

References

[1] ASTRA, *Annex i, description of work* (Contract for specific targeted research or innovation project, September 2005).

[2] L.-F. Cabrera and C. Kurt, *Web services architecture and its specifications: Essentials for understanding ws-** (Microsoft Press, Redmond, WA, USA, March 2005).

[3] W. K. Edwards, V. Bellotti, A. K. Dey, and M. W. Newman, 'The challenges of user-centered design and evaluation for infrastructure', in *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems* (ACM Press, New York, NY, USA, 2003), pp. 297–304.

[4] C. A. Ellis, S. J. Gibbs, and G. Rein, 'Groupware: some issues and experiences', *Commun. ACM* **34** (1991), no. 1, pp. 39–58.

[5] B. A. Farshchian and M. Divitini, *Ubicollab - white paper* (IDI/NTNU Technical Report. Published on the UbiCollab web site, June 2007). `http://mediawiki.idi.ntnu.no/wiki/ubicollab/`.

[6] K. Finkenzeller, *Rfid handbook: Fundamentals and applications in contactless smart cards and identification* (John Wiley & Sons, Inc., New York, NY, USA, 2003).

[7] S. Haller and S. Hodges, *Auto-id centre white paper: The need for a universal smartsensor network* (Published online, available from the Auto-ID Labs homepage, November 1st 2002). `http://www.autoidlabs.org/uploads/media/CAM-AUTOID-WH-007.pdf`.

[8] S. Hodges and D. McFarlane, *Radio frequency identification: technology, applications and impact* (Published by Auto-ID Lab, Cambridge University, UK. Downloadable from www.autoidlabs.org, September 2005). `http://www.autoidlabs.org/uploads/media/`.

[9] T. Iso, S. Kurakake, and T. Sugimura, 'Visual-tag reader: Image capture by cell phone camera', in *Proceedings of the 2003 International Conference*

on Image Processing (ICIP 2003), Barcelona, Catalonia, Spain, **2** (IEEE Computer Society, Washington, DC, USA, September 2003), pp. 557–560.

[10] K. S. Johansen and M. A. Børke, *User friendly and community oriented service discovery in a pervasive awareness system* (Depth study, Norwegian University of Science and Technology, Trondheim, Norway, Autumn 2006).

[11] A. Juels, R. L. Rivest, and M. Szydlo, 'The blocker tag: selective blocking of rfid tags for consumer privacy', in *Proceedings of the 10th ACM conference on Computer and communications security (CCS '03)* (ACM Press, New York, NY, USA, 2003), pp. 103–111.

[12] T. Kindberg, J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, J. Schettino, B. Serra, and M. Spasojevic, 'People, places, things: Web presence for the real world', *Mobile Networks and Applications* **7** (2002), no. 5, pp. 365–376.

[13] C. D. Knutson and J. M. Brown, *Irda principles and protocols: The irda library*, **1** (MCL Press, May 2004).

[14] J. Landt, 'The history of rfid', *Potentials, IEEE* **24** (October-November 2005), no. 4, pp. 8–11. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1549751`.

[15] C. Laverton, *Social tagging of services to support end user development in ubiquitous collaborative environments* (Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway, Spring 2007).

[16] C. H. Mosveen and A. Brustad, *Ubicollab: A service architecture for supporting ubiquitous collaboration* (Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway, Spring 2006).

[17] H. Ogata and Y. Yano, 'Context-aware support for computer-supported ubiquitous learning', in *Proceedings of the 2nd IEEE International Workshop on Wireless and Mobile Technologies in Education (WMTE'04)* (IEEE Computer society, 2004), pp. 27–34.

[18] C. Schwarz, *Ubicollab - platform for supporting collaboration in a ubiquitous computing environment* (Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway, Spring 2004).

[19] M. L. Segelvik and E. L. Segelvik, *Location awareness in ubicollab* (Master's thesis, NTNU, Spring 2007).

[20] S. Siltanen and J. Hyväkkä, 'Implementing a natural user interface for camera phones using visual tags', in *AUIC '06: Proceedings of the 7th Australasian User interface conference* (Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 2006), 113–116.

[21] N. Varchaver, 'Scanning the globe', *Fortune Magazine* **149** (2004), no. 11, pp. 144–148.

[22] P. Välkkynen, M. Niemelä, and T. Tuomisto, 'Evaluating touching and pointing with a mobile terminal for physical browsing', in *Proceedings of the 4th Nordic conference on Human-computer interaction (NordiCHI '06)* (ACM Press, New York, NY, USA, 2006), pp. 28–37.

[23] M. Weiser, 'The computer for the 21st century', *Scientific American* (1991), pp. 94–104.

Appendix A

Task description

This appendix presents the original task description, and the final one. The original task description has been updated somewhat during the semester and changed from Norwegian to English. The original task description was specified in Norwegian because of an inadvertency.

A.1 Original task description

Brukersentrert og gruppebasert tjenestepublisering og oppdagelse i UbiCollab

Tjenesteoppdagelse (eng: service discovery) i UbiCollab tar for seg hvordan tjenester dynamisk kan oppdages i brukerens miljø. Mens tjenesteoppdagelse tradisjonellt har fokusert på maskin-maskin interaksjon har UbiCollab en helt annen tilnærming ved å la brukerne selv styre publisering, oppdagelse og deling av tjenester. Tjenesteoppdagelse er en kjernekomponent i UbiCollab og i allstedsnærværende samhandlingsteknologi (eng: ubiquitous computing) generelt.

Denne oppgaven skal søke å finne en løsning for administraring av tjenester for UbiCollab plattformen. Den enkelte bruker skal være i stand til å administrere tjenestene på sine UbiCollab noder og utføre oppgaver som installasjon, publisering, fjerning og overvåking av tjenester. Leveranser for oppgaven skal bestå av en rapport basert på strukturen for UbiCollab ingeniørprosjekter, samt implementasjon av løsningen. Dette prosjektet er en del av UbiCollab. See http://research.idi.ntnu.no/ubicollab

A.2 Final task description

User-centered and collaborative service management in UbiCollab: Design and implementation

UbiCollab is a platform for supporting ubiquitous collaboration between people.

UbiCollab is being developed in cooperation between the Norwegian University of Science and Technology, Department of Computer and Information Science, and Telenor Research & Innovation. The UbiCollab project is open source, and interacts with a number of European projects and with industrial partners internationally.

This task is related to service discovery and management, which is an important part of the UbiCollab's service oriented architecture. The candidate will propose and implement a solution for a user-centered and collaborative service management system for UbiCollab. The solution proposal will be founded on a previous study on this subject where a requirement specification and high level architecture were proposed.

Appendix B

Functional Requirements

In this appendix the functional requirements derived during the autumn project [10] is presented. Some of the wording have also been rephrased because of differences in vocabulary and concepts between UbiCollab and ASTRA. The requirements are listed in tables and grouped according to which functionality they are closest related to. Each requirement is tagged by IDs named FR-Yx, where Y is a letter separating the categories of requirements, and x is the number of the requirement (and FR is an abbreviation for Functional Requirement). This will improve the traceability of the requirements.

Individual priority-assignment for each requirement have also been performed. Each requirement is weighted by High (**H**), Medium (**M**) or Low (**L**) importance, to help prioritize if any requirements are in conflict or not reasonable[1] to solve. In addition, the anticipated degree of difficulty (abbreviated as DoD) is given to better understand the extent of implementing each requirement. This will be weighted the same way as importance, where high means "high difficulty" and low means "low difficulty".

In addition to this, common service management tasks were investigated in the autumn report but not considered a part of the service discovery solution. Due to this functional requirements for management were not elicited. Since this project need these requirements, they have been elicited as part of this project. These requirements have been listed in a table like the others, and can be found in Table B.4

Moreover, a set of infrastructure related requirements were elicited in the autumn report. These requirement were directed toward the overall architecture for ASTRA, and covered the assumptions that had to be made about other parts of the system. In UbiCollab these are not needed because the necessary architectural choices have already been made. Therefore the infrastructure related requirements have been removed, except for one. This requirement have been renamed from "FR-I7" to "FR-O11" and can be found in Table B.5.

[1]Some requirements may take to much time to solve, related to how important they are. If this is the case, it is assumed to be unreasonable to include them in the solution.

A few additional requirements have also been elicited because of the specific support for physical discovery and the generic support for other service discovery protocols. These have been added to the functionality group in Table B.2.

Table B.1: Functional Requirements: Service publishing

ID	**Description**	**DoD**	**Priority**
FR-P1	The module must provide a Web service interface for service publishing	H	H
FR-P2	A service must be published to be available for other users	L	H
FR-P3	The publishing process must be user initiated, and possible to cancel on user request	M	H
FR-P5	The module must require that mandatory parts of the service description are correctly filled in before the service is published	M	M
FR-P6	The technical details of the registration process should be as transparent to the user as possible	M	M
FR-P7	It must be possible to unpublish services	M	M

Table B.2: Functional Requirements: Service search

ID	**Description**	**DoD**	**Priority**
FR-S1	The solution must provide a Web service interface for service search	H	H
FR-S2	The solution must aid the user in the service search	H	H
FR-S3	Services must be discoverable across logical and geographical borders	H	H
FR-S4	A user must not be allowed to discover a service which is not visible to him or her	M	H
FR-S5	Service discovery must support both proxy services and self contained services	L	H
FR-S6	The returned service descriptions must contain information about the service state	M	M
FR-S7	The progress of the search must be communicated to the user	H	M
FR-S8	The results of a search should be ranked by relevance	H	L
FR-S9	The solution should log events related to service discovery operations	M	M
FR-S10	The solution should support physical service discovery	H	H
FR-S11	The solution should provide a generic interface for service discovery protocols	H	H

Table B.3: Functional Requirements: Service description

ID	**Description**	**DoD**	**Priority**
FR-D1	Services must be described by searchable attributes	L	H
FR-D2	Service descriptions must conform to one common specification	H	H
FR-D4	User specific information must be typed into the service description by the owner of the service	L	M
FR-D7	Context information (like location) must be used in the service description when relevant	M	M
FR-D8	Context attributes in the service description must be dynamic, and should be updated when the context changes	H	M
FR-D9	As much information as possible should be predefined in service descriptions	M	L
FR-D10	Service discovery matching must happen based on user-friendly service description attributes	M	L
FR-D11	Services residing in a service domain should be described with UbiCollab space properties	L	M

Table B.4: Functional Requirements: Service Management

ID	**Description**	**DoD**	**Priority**
FR-M1	It must be possible to install services	M	H
FR-M2	It must be possible to start and stop services	L	H
FR-M3	It must be possible to edit service properties	M	M
FR-M4	Service properties must be stored persistently	M	M
FR-M5	It must be possible to retrieve the registered information about one or more services	M	M
FR-M6	It must be possible to retrieve the state information about a service	M	M
FR-M7	It must be possible to uninstall services	L	M

Table B.5: Functional Requirements: Other

ID	**Description**	**DoD**	**Priority**
FR-O1	It must be possible to use the service discovery and management solution on a limited device, such as a PDA	M	H
FR-O2	The user must be given an overview of which services are registered in the service domain, and who have access to them	M	H
FR-O3	The dynamic properties of the registered service must be changeable at all times	H	H
FR-O4	The solution must provide access to service information like state, visibility and user access	M	H
FR-O5	The service must be automatically unpublished when the service is uninstalled	M	M
FR-O6	There should be a log system for service discovery and management events	L	M
FR-O7	It must be possible to lock a service for one user at a time	M	M
FR-O8	All service contract negotiations and their results must be logged	M	L
FR-O9	The solution must provide a interface where services can update their states	H	L
FR-O10	The owner of a service must be able to break the contract at any time, disabling a service possibly in use	M	L
FR-O11	Commercial actors must be allowed to provide services	M	M
FR-O12	It must be possible to retrieve logged information	M	L

Appendix C

Application programming interfaces

This appendix will provide a detailed description of the APIs for the components in the service management solution.

C.1 Service Domain Manager API

getServiceList()

Return a list of the services which have been installed on this UbiNode through the Service Domain Manager. The method returns an XML-formatted `String` with all installed services. The xml-structure used to represent the returned service list is the format for advertised services, with additional attributes as described in Section 4.2.1.

getInfoAboutService(Long bundleID)

Gets all registered information about the specified service. The xml-structure used to represent the returned service is the format for advertised services, with additional attributes as described in Section 4.2.1. "INVALID BUNDLE ID" will be returned if an invalid bundle id is provided.

Parameters:

- bundleID - the bundle ID the bundle is registered with.

installService(String url, String friendlyName)

Installs a service (bindle). Takes the URL to a service bundle and the new service name as parameters. The name of the service is optional, the default name defined in the bundle will be used if none is provides. Will return a human readable text describing the outcome of the installation.

Parameters:

- friendlyName - the friendly name of the service.
- url - The Url to the service bundle.

checkIfServiceExist(String uri)

Checks if a given service exist. Takes the service URI as parameter and uses this to determine if the service exist. The service URI is unique for each service in the domain. Returns `true` if the service exists, `false` otherwise.

Parameters:

- uri - the service uri

removeService(Long bundleID)

Removes the service proxy which is specified by the given bundleID. The method will return `true` if the service exist and are successfully removed, `false` otherwise.

Parameters:

- bundleID - The URI for the service which is to be removed

updateServiceProperties(parameter list - see below)

Method to update the properties for the service identified by the bundle ID. Will return `true` if the bundle id is found and update is OK, `false` otherwise.

Parameters:

- bundleID - the bundle ID for the service to update
- friendlyName - The service friendly name
- description - the textual description of the service
- space - The description for the space this service will be active in
- spaceId - The space id for the space this service will be active in

startService(Long bundleID)

Method to start the installed service identified by the provided bundle ID. Returns `true` if the service is successfully started, `false` otherwise.

Parameters:

- bundleID - the bundle ID for the bundle providing the service

stopService(Long bundleID)

Method to stop the installed service identified by the provided bundle ID. Returns `true` if the service is successfully stopped, `false` otherwise.

Parameters:

- bundleID - the bundle ID for the bundle providing the service

getLogItems(int logLevel, int numItems)

Read from the log and return selected log items in a XML-formatted string. The XML-Schema the returned list of log-elements will conform to, have been described in Figure 4.7

Parameters:

- logLevel - The log level to include in the report. The valid values are:
  - 0="DEBUG"
  - 1="INFO"
  - 2="WARNING"
  - 3="ERROR"
  - 4="FATAL"
- numItems - The number of log items to include in the returned list

setActiveSpace(int SpaceID, boolean stopOtherServices)

Activates all services belonging to the space identified by the provided id. Also provides the option to stop the other services. The method will return `true`, if all proxies adhering to the space ID is started, `false` otherwise.

Parameters:

- SpaceID - the ID for the Space to activate
- stopOtherServices - true to stop the other service proxies, false othervise

C.2 Service Discovery Manager Plugin API

registerSDPlugin(String protocolName)

This method will register a Service Discovery plugin with the Service Discovery Manager. When a plugin is registered a unique ID is returned, which will be used to identify the plugin in further communication. The registration will have a lifespan of 30 seconds, and will be reset to 30 seconds every time `getClien-tRequest` or `registerService` is invoked. When this time runs out the plugin session will be garbage collected (by the housekeeper thread).

Parameters:

- `protocolName` - the name of the protocol, e.g. UPnP or RFID

getClientRequest(int pluginID)

This method will be used by plugins to fetch the ClientRequests registered by client(s). The returned list of clientrequests will be formated according to the XML Schema in Figure 4.13.

Parameters:

- `pluginID` - the plugins identifier (as issued by Discovery Manager)

registerService(parameter list - see below)

This method is provided for service discovery plugins to register services which matches client requests, with the Service Discovery Manager. The service will be added to the queue identified by the provided searchRequestID. This method provides many options as parameters and all of these will not be applicable for all plugins. The ones not used should simply contain an empty string.

Parameters:

- `friendlyName` - The user-friendly name of the service
- `descriptionURI` - URI for service description (e.g. the URL to a WSDL-file)
- `type` - The service type (classification)
- `owner` - The person/company/responsible for the service
- `description` - Tags (words or short sentences) describing the service
- `location` - Location where the service can be found
- `searchRequestorID` - the ID for the search request queue to add this service to
- `protocolID` - The id for the service discovery protocol plugin
- `serviceURI` - The URI to invoke the service

C.3 Service Discovery Manager Client API

registerClient()

This method allows any authenticated user (e.g. client application) to register as a client with the Service Discovery Manager (SDM). When a client is registered a service queue is created and assigned to him. Upon registration the client will also be assigned a unique ID which will be used for identification when sending messages/communicating with the Discovery Manager. The unique ID to be used for identification when communicating with the Discovery Manager will be the return parameter.

getServices(int clientID)

Return the found services from the list identified by the provided search request ID. After the results are returned they will be removed from the queue. The XML-Schema for the returned service list is displayed in Figure 4.5.

Parameters:

- `clientID` - the clients identifier (as issued by Discovery Manager)

getSDPlugins()

Method to return a list containing the plugins registered with the Discovery Manager. Useful method for clients who want to know which service discovery protocols they have support for. Returns a XML-formatted list with the plugins, conforming to the XML-Schema shown in Figure 4.11.

registerClientRequest(parameter list - see below)

This method allows any authenticated user (client) to register a search request with the Service Discovery Manager. The manager will then make this request available to all registered plugins. All the parameters provided will be used for matching against advertisements.

Parameters:

- `owner` - the Owner of the service to search for
- `type` - the service type to search for
- `description` - the description of the service
- `location` - the location of the service
- `clientID` - the unique client ID for the client issuing the request. When referenced by plugins they will name this id as `requesterID`.
- `name` - the name of the service to search for

getLogItems(int logLevel, int numItems)

Read from the log and return selected log items in a XML-formatted string. The XML-Schema the returned list of log-elements will conform to, have been described in Figure 4.7

Parameters:

- logLevel - The log level to include in the report. Valid arguments are:
  - 0="DEBUG"
  - 1="INFO"
  - 2="WARNING"
  - 3="ERROR"
  - 4="FATAL"
- numItems - The number of log items to include in the returned list

C.4 Service Registry API

getServices()

This method returns an XML-formatted list containing all the services in the registry. The returned XML-document will conform to the XML-Schema for service lists displayed in Figure 4.5.

getServices(parameter list - see below)

Method to return a list of service adverticements matching the criteria specified in the query. The different search criteria can be freely combined and also partial hits will be returned. Criteria not used should contain an empty string or the `null` property. The returned XML-document will conform to the XML-Schema for service lists displayed in Figure 4.5.

Parameters:

- `type` - The type of service, or a partial type specification
- `owner` - The the owner of the service, or a part of the owners name
- `description` - The textual description of the service
- `location` - The location for the service, or a part of the location
- `name` - The name of the service, or a part of it

getServiceDetails(String serviceUri)

Gets all registered information about the service specified by the provided `serviceUri`. The returned XML-document will conform to the XML-Schema for services displayed in Figure 4.4.

Parameters:

- `serviceUri` - The service URI that uniqly identifies the service

getServiceDetails(int serviceId)

Gets all registered information about the service specified by the provided `serviceUri`. The returned XML-document will conform to the XML-Schema for services displayed in Figure 4.4.

Parameters:

- `serviceId` - The service Id that uniqly identifies the service

addService(parameter list - see below)

Adds a service advertisement to the registry. This method can be used to add new services which can be found by other users. The method returns true if the service advertisement is successfully updated, false otherwise.

Parameters:

- `type` - The type of service
- `owner` - The the owner of the service
- `description` - A textual description of the service
- `location` - The location for the service
- `name` - The name of the service
- `descriptionURI` - The URI pointing to the service description for the service
- `serviceURI` - The URI where the service can be contacted, or found

updateService(parameter list - see below)

Updates the properties of a given service advertisement in the service registry. The key used to select which service to update is the `serviceID`. The method returns true if the service advertisement is successfully updated, false otherwise.

Parameters:

- `serviceId` - The service Id that uniqly identifies the service
- `type` - The type of service

- `owner` - The the owner of the service
- `description` - A textual description of the service
- `location` - The location for the service
- `name` - The name of the service
- `descriptionURI` - The URI pointing to the service description for the service
- `serviceURI` - The URI where the service can be contacted, or found

deleteService(int serviceId)

Removes the service advertisement which matches the provided `serviceId` parameter.

Parameters:

- `serviceId` - The service Id that uniqly identifies the service

getLogItems(int logLevel, int numItems)

Read from the log and return selected log items in a XML-formatted string. The XML-Schema the returned list of log-elements will conform to, have been described in Figure 4.7

Parameters:

- logLevel - The log level to include in the report. The valid values are:
  - 0="DEBUG"
  - 1="INFO"
  - 2="WARNING"
  - 3="ERROR"
  - 4="FATAL"
- numItems - The number of log items to include in the returned list

C.5 X10 demonstrator API

stopSerialConnection()

Stops the ongoing serial connection and releases resources in use.

startSerialConnection(String comPort)

Start a serial connection to a CM11A serial X10 controller, on the provided COM port.

Parameters:

- `comPort` - the serial communication port to use, must be adapted to the platform running the proxy. E.g. “COM1”, “COM5” or “/dev/tty/s2”.

sendCommand(String house, int unit, boolean turnOn)

Method to send an X10 on/off command to a device identified by the given address (`house` + `unit`). This method assumes that a serial connection to a cm11a X10 device already have been started. Nothing will happen if a serial connection have not been opened.

Parameters:

- `unit` - The unit address, valid range are 1..16
- `house` - The house address, valid range are A..P
- `turnOn` - `true` to turn the power on, `false` to turn the power off

dimDevice(String house, int unit, int cmnd, int percentage)

Method to send dim or brighten commands to the X10 device identified by the given address (`house` + `unit`). This method assumes that a serial connection to a cm11a X10 device already have been started. Nothing will happen if a serial connection have not been opened.

Parameters:

- `cmnd` - Valid values are 0 for dim or 1 for brighten
- `unit` - The unit address, valid range are 1..16
- `house` - The house address, valid range are A..P
- `percentage` - the level to change, valid range between 1 and 100 (percentage)

Appendix D

Gestalt Principles

The Gestalt Principles refer to theories of visual perception. These theories are widely used in the HMI (Human Machine Interface) field, and attempts to describe how people tend to organize visual elements into groups or unified wholes when certain principles are applied. In this appendix some of the principles which must be considered while planning the user interface will be described.

D.1 Similarity

The principle of similarity states that things which share visual characteristics such as shape, size, color, texture, value or orientation will be seen as belonging together. People will often perceive them as a group or pattern.

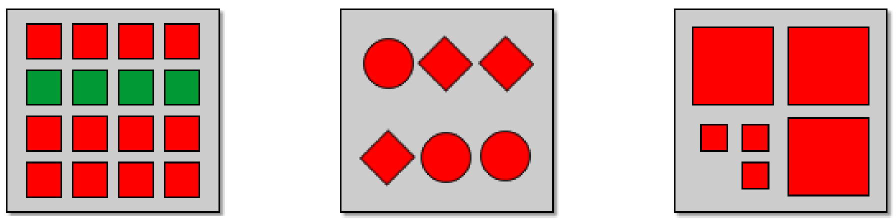


Figure D.1: Similarity in color, form and size

Looking at the leftmost figure displayed in Figure D.1, we see that the green boxes give the impression of a horizontal line, and that they somehow seem to be grouped together or have something in common although all the boxes are equidistant from each other.

From the other two figures we can see that similarities in form and size also gives a sense of belonging together. From the second figure displayed we can also see that a shape which is different from the other can give a sense that it is not a part of a group.

D.2 Proximity and Continuity

The principle of proximity states that things which are close together will be seen as belonging together. The closer two or more visual elements are, the greater the probability they will be seen as a group or pattern. In the leftmost figure in Figure D.2 the elements which are close together seem to form a group, even though all the boxes are equal in size and color.

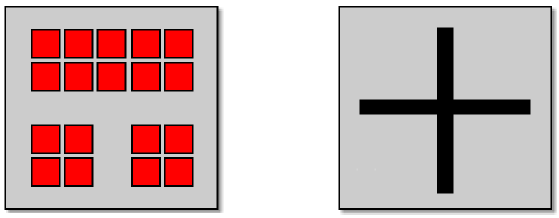


Figure D.2: Proximity and Continuity

The principle of continuity predicts the preference for continuous figures. When we look at the second figure shown in Figure D.2 we perceive the figure as two crossed lines instead of 4 lines meeting at the center.

D.3 Symmetry

The principle of symmetry describes the instance where the whole of a figure is perceived rather than the individual parts which make up the figure.

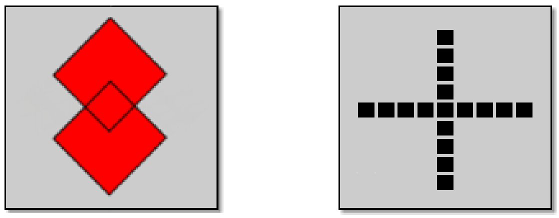


Figure D.3: Symmetry between elements

In the first of the figures displayed in D.3 you can either see two overlapping diamonds, or three objects (a small diamond and two irregular objects above and below it). If you are perceiving according to the principle of symmetry, you will probably see two diamonds overlapping. In the second figure you will most likely see a cross first, instead of a group of equal sized boxes.

Appendix E

Overview of the IDBlue RFID Device

IDBlue is a 13.56 MHz RFID reader, in the shape of a pen with dimension 112mm x 25mm x 16mm and a weight of 50 grams. It uses Bluetooth technology to wireless link up with most Bluetooth compliant devices, such as PC's, PocketPC's and SmartPhones. This appendix is intended to provide a quick introduction to the features and capabilities of the IDBlue device.

E.1 Device Aspects

- RFID reader for near-field applications, typical 20-40mm reading distance.
- Support Bluetooth Bluetooth® 1.1, Class 2. Maximum range approximately 15m.
- Requires Microsoft or Widcomm Bluetooth protocol stack.
- Manufacturer states Microsoft Windows 2000/XP or Microsoft Windows Mobile w/Bluetooth Support as requirement, but states that the device have been deployed successfully on other platforms as well.
- The device can operating both connected to a PC and as a standalone unit (capable of storing up to 1000 tags).
- Supports the following RFID operations:

  - Select tag ID
  - Read data
  - Write data

- .NET driver support for .NET Framework 1.1 and compatibility with 2.0 (including Compact Framework) is provided and supported by manufacturer.

In addition, documentation of interest for development purposes are provided by the manufacturer[1]. This documentation includes:

- Developer's Guide for .NET
- API Reference Guide for .NET
- Technical Specification
- Serial Protocol Definition

E.2 LED States

Table E.1 lists the various LED states that provide feedback to the user as to the current operating state of the IDBlue device. The IDBlue's LED is often the only feedback provided by the pen, and as such it provides valuable information for development.

Table E.1: LED states for IDBlue RFID Pen

LED Behavior	**Connection Status**	**Battery Status**	**RFID Operation**
Green, Single-flash	Unconnected	Battery Ok	None
Green, Double-flash	Connected	Battery Ok	None
Red, Single-flash	Unconnected	Battery Low	None
Red, Double-flash	Connected	Battery Low	None
Amber, Solid	N/A	N/A	In progress
Red, Solid	N/A	N/A	Failed
Green, Solid	N/A	N/A	Success
Red, Rapid flash	Device needs Firmware Update.		

[1]http://www.cathexis.com/support/downloads.aspx (Visited 20.05.2007)

Appendix F

Installing the Java Communications API

In this appendix a short description of the operations needed to install the Java Communications API Version 2.0 on a Microsoft Windows operating system will be provided. The problem related to this package is that the developer, Sun Microsystems[1], does not support this product for Microsoft Windows since 2002. The product has not been found available for download from their Internet site either, so it must be downloaded from elsewhere. This seems to be a common problem with this package. However, newer versions have been developed for the Solaris and Linux operating systems and are available for download from Sun.

Two sites where the Microsoft Windows version of the product can be downloaded are:

- `http://www.tip.no/tini/javacomm20-win32.zip` (Visited 10.06.2007)
- `http://www.oreilly.com.tw/bookcode/java_io/` (Visited 10.06.2007)

In the following it is assumed that the Java version in use is Sun JDK 1.6.0. If another Java version is being used, this must be made taken into consideration. The steps needed for installation are:

1. Download the `javacomm20-win32.zip` package
2. Unpack the files to `C:\Programfiler\Java\jdk1.6.0`
3. Copy `win32com.dll` from `C:\Programfiler\Java\jdk1.6.0\commapi` to `C:\Programfiler\Java\jdk1.6.0\bin`
4. Copy `javax.comm.properties` from `C:\Programfiler\Java\jdk1.6.0\commapi` to `C:\Programfiler\Java\jdk1.6.0\jre\lib`

[1]http://java.sun.com/products/javacomm/index.jsp (Visited 10.06.2007)

5. Copy `comm.jar` from `C:\Programfiler\Java\jdk1.6.0\commapi` to `C:\Programfiler\Java\jdk1.6.0\jre\lib\ext`

6. That should be it. The Java comm API is now installed.

Appendix G

Dependencies

This appendix describes the dependencies for components in the service management solution.

G.1 Service Registry

The Service Registry depends on a MySQL database being installed. Default connection settings are set to:

Name: “ubicollab”
Username: “ubi”
Password: “ubi4ALL”
Table: “serviceregistry”

The following SQL script will remove any existing table with the name “serviceregistry”, and generate a new table with the correct fields:

```
DROP TABLE IF EXISTS ‘ubicollab‘.‘serviceregistry‘;
CREATE TABLE  ‘ubicollab‘.‘serviceregistry‘ (
‘ID‘ int(10) unsigned NOT NULL auto_increment,
‘ServiceURI‘ varchar(145) NOT NULL,
‘DescriptionURI‘ varchar(145) default NULL,
‘Name‘ varchar(45) default NULL,
‘Type‘ varchar(80) default NULL,
‘Location‘ varchar(80) default NULL,
‘Owner‘ varchar(45) default NULL,
‘Description‘ varchar(145) default NULL,
PRIMARY KEY  (‘ID‘)) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

G.2 Service Discovery Plugin for RFID

This plugin requires that the following package is installed on the PC or PocketPC which will be running the plugin:

Windows PC Franson Bluetools for PC

PocketPC Franson Bluetools for ARM, x86 PocketPC

The required assemblies (DLL files) from these packages can be extracted and included in the folder where the Plugin runs, in order to provide a more lightweight distribution. It should however be noted that the Franson Bluetools bundled with the IDBlue RFID pen is not freely redistributable.

The complete list of assemblies required for different platforms and different .NET versions are described in the "IDBlue.NET Developer Guide" that comes bundled with the IDBlue pen (or can be downloaded from the Cathexis Innovations website[1]).

[1]http://www.cathexis.com (Visited 13.06.2007)