NTNU
Innovation and Creativity

Social Tagging of Services to Support End User Development in Ubiquitous Collaborative Environments

Christian Laverton

Master of Science in Computer Science
Submission date: June 2007
Supervisor: Monica Divitini, IDI

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

Problem Description

The task is within the context of Ubicollab, a platform for the development of ubiquitous collaborative applications, and ASTRA, a European project for supporting end-user development of pervasive awareness applications, i.e. applications that help the members of a distributed community to feel connected. The task focuses on end user development in terms of support to end users in selecting the right services to compose in order to create their applications. Different languages have been proposed for identifying Web services and then compose them. Standards for web service descriptions are also available. However, the resulting descriptions are generally not intended nor understandable to end-users. The task investigates the usage of social tagging for supporting a community of users in associating a meaning to services that they discover in the spaces that they inhabit. The task aims at extending Ubicollab to support tagging of services.

Assignment given: 20. January 2007
Supervisor: Monica Divitini, IDI

Abstract

Tailorability in ubiquitous computing systems is needed at different levels, depending on the targeted end users. For inexperienced end users lacking computer competency, high level mechanisms for tailoring are needed. Systems such as Awareness Services and Systems - Towards theory and ReAlization (ASTRA), which use a Service Oriented Architecture (SOA), can provide such high level tailorability through service composition. With service composition, services can be combined and configured to form applications.

However, using service composition introduces new challenges for end users. To find appropriate services, users need mechanisms for searching and browsing services. Equally important is it that users are able to understand how services work and what functionality they offer. Service descriptions can ease this task, but the problem with existing approaches to service descriptions is that they are not intended for end users and are hard to understand.

This work looks at social tagging, which is a collaborative process where users attach labels or tags to items. This leads to user created metadata, as opposed to metadata created by experts. By introducing social tagging in ASTRA to describe services, users are provided with a framework for sharing their understanding of services with fellow users.

To create a solution for social tagging for service descriptions, a thorough problem analysis was performed. The analysis considered the design space of tagging systems to find appropriate design choices in the problem context. Providing several tag visibility levels was identified as important, especially community tagging. The quality of tags as seen from the community members' perspective is likely to increase, as members of communities often share similar opinions and understandings. An important difference identified between existing tagging systems and tagging of services is that services can be embedded in physical devices. Thus, services can be discovered and accessed physically, which means that physical access to the services' tags should be supported.

A requirements specification for a tagging system was specified, focusing on the platform requirements for basic tagging mechanisms, tag based navigation, and searching. The requirements lead to a design of platform architecture, aiming at extending the UbiCollab platform with social tagging functionality. The architecture uses a client/server solution, where the server service is shared among a network of users and handles public and community level tags. The client service is a local service which handles private tags, and acts as an intermediary between end user tools and the server service. A prototype of the platform services and an end user tool was implemented. The implementation is demonstrated through scenarios, showing possible uses of the tagging system.

Keywords: social tagging, folksonomy, services, service composition, ubiquitous computing, ASTRA, UbiCollab

Preface

This report was written for the work done as a Master's thesis at the Department of Computer and Information Science (IDI), Norwegian University of Science and Technology (NTNU). The work was done during spring 2006.

The task description for the thesis was the following:

> *The task is within the context of Ubicollab, a platform for the development of ubiquitous collaborative applications, and ASTRA, a European project for supporting end-user development of pervasive awareness applications, i.e. applications that help the members of a distributed community to feel connected. The task focuses on end user development in terms of support to end users in selecting the right services to compose in order to create their applications. Different languages have been proposed for identifying Web services and then compose them. Standards for web service descriptions are also available. However, the resulting descriptions are generally not intended nor understandable to end-users. The task investigates the usage of social tagging for supporting a community of users in associating a meaning to services that they discover in the spaces that they inhabit. The task aims at extending Ubicollab to support tagging of services.*

I would like to thank my supervisor, Professor Monica Divitini. Also thanks to Professor Babak Farshchian and Kim Steve Johansen for valuable input regarding UbiCollab.

Trondheim, June 17th, 2007

________________-

Christian Laverton

Contents

List of Figures

List of Tables

Acronyms

API	Application Programming Interface
ASTRA	Awareness Services and Systems - Towards theory and ReAlization
CI	Collaboration Instance
CS	Collaboration Space
CSCW	Computer Supported Cooperative Work
ER	Entity Relationship
GUI	Graphical User Interface
IDI	Department of Computer and Information Science
NTNU	Norwegian University of Science and Technology
P2P	Peer-to-peer
PDA	Personal Digital Assistant
RFID	Radio Frequency Identification
SOA	Service Oriented Architecture
SP	Service Proxy
SD	Service Domain
SDM	Service Discovery Manager
SR	Service Registry
TM	Tag Manager
UDDI	Universal Description, Discovery and Integration
UPnP	Universal Plug and Play
URI	Uniform Resource Identifier
XML	Extensible Markup Language

Chapter 1

Introduction

Ubiquitous computing systems are largely characterised by change and diversity. A similar characterisation can be given of the users. Different levels of competency and different tasks and interests are only some of the factors that lead to users having different requirements to systems. This means that it is hard or even impossible for developers to create systems that fit all users' needs, and foresee future requirements. This has lead to a need for systems to be changeable, or tailorable, by users. Tailoring means to modify the behaviour of the system, and usually involves more radical changes than the simple changing of options and parameters. These changes must be possible to do without programming knowledge [8].

1.1 Services and service composition

The need for tailorability in ubiquitous computing has lead to the need for tools supporting end user development and tailoring. An important requirement for such tools is that different levels of tailorability are provided to support different groups of users [1]. End users could be experienced programmers, thus requiring low level mechanisms for developing applications. But in many cases end users will not have the required competency, which means higher level mechanisms are needed.

A ubiquitous computing system is filled with a multitude of services of various kinds. A service can be a physical device with capabilities such as audio, video, light and movement, or an application that does not reside in the physical world. In any case, the service provides some sort of functionality, able to support the user in performing a certain task. By adopting a SOA, a system can provide such high level mechanisms for end user development, by enabling end users to compose applications from a selection of services. This is called service composition.

1.2 Describing services

Given tools for service composition, the user's challenge is to find and select appropriate services. With the large amount of services and diversity in service types and usage, this is not a trivial task. To make it easier, one of the most important challenges is how services can be described, in a way suitable for the user.

The need for service descriptions first arises when the user wants to search for appropriate services. The number of services is often too large for the user to look through each and every one, so filtering mechanisms for limiting the number of candidates are obviously needed. Here, it is natural for the user to supply a text string or a list of keywords, indicating service characteristics and functionality. In other words text that he would expect an appropriate service's description to contain.

In addition to finding services, users need to understand them. Without knowing what functionality a service offers, where it is located and if it is accessible, users would have no way of knowing if the service is worth using. Traditionally, software and hardware have been described by the manufacturers, by providing a specification and description of why users should choose their product. A similar approach could be used to describe services, with service creators or owners publishing services accompanied by specifications, textual descriptions, or similar. Also, formal languages such as ontologies are commonly used to describe and classify services. But formal languages are complex and difficult to use, which means they are best suited for expert use. A problem with such expert descriptions or classifications is that there is often quite a signficant gap between the experts' and the users' views and understandings. This can be referred to as a semantic gap [15].

A solution to filling the semantic gap could be to let users collaboratively describe services, thus helping each other to increase their understanding and choose the right services.

1.3 Research context

The research context of this work is ASTRA and UbiCollab. The problem elaboration, analysis and requirements specification focus on the problem in relation to the ubiquitous computing system ASTRA. In the design and implementation phases however, the work will be based on UbiCollab. This means that the design proposed here, as well as the actual implementation, will be part of the UbiCollab system. The reason for this transition is that UbiCollab is currently being used as a starting point for the development of the ASTRA platform [11].

1.3.1 ASTRA

The ASTRA project is aimed at researching the concept and theory of pervasive awareness. Awareness systems, as described in [1] are systems that help individuals or groups to build and maintain peripheral awareness of each other. Further, pervasive awareness is where awareness information is generated from personal and home devices, which capture and exchange information about users.

While quite a lot of studies have been performed in work-related contexts, very little has been done in social contexts. Awareness in the social context, or interpersonal awareness, is characterised as the understanding of the activities and status of one's social relations. The need to stay in touch or the need to be assured of other's well-being are both examples of interpersonal awareness outside the work-context. ASTRA is not aimed at individuals, but communities. The intent is to support communication within a community, to increase the feeling of connectedness between members of the community.

ASTRA aims to contribute to our knowledge in this area through a theory and supporting technology for creating pervasive awareness systems.

1.3.2 UbiCollab

UbiCollab is a platform for supporting collaboration on the internet. The main idea of the platform is to combine mobility and ubiquity with traditional Computer Supported Cooperative Work (CSCW) and groupware technologies. In this way, UbiCollab attempts to naturally support collaboration in any situation the users are in [3]. This means that UbiCollab supports collaboration in general, while ASTRA can be seen as a specific use of UbiCollab, mainly concerned with communities.

Similar to ASTRA, the UbiCollab platform is meant to capture common functionality in collaborative applications. The common functionality provided by the platform can be used to build applications without extensive coding [3].

The UbiCollab platform has been partially implemented, and improved in several newer versions. This means that UbiCollab provides an architecture and code which can be used in this work's design and implementation.

1.4 Project goals

This work aims to support users in understanding services as well as search for and find appropriate services in a ubiquitous computing environment. These are important criteria for the success of service composition, a mechanism for end user development based on SOAs. To accomplish this, the focus will be on how users can share user-generated service descriptions and information regarding services in a collaborative manner, using social tagging. Thus, the main goal of this project is to design and implement a framework for social tagging of services in a ubiquitous computing context. This framework will be designed in form of services and tools to extend the UbiCollab platform.

1.5 Contributions

This project presents the following contributions:

- Problem analysis
- Requirements specification
- Design of platform architecture
- Platform service APIs
- Implementation of platform services
- Demonstrator tool

In the *problem analysis*, important design issues for tagging systems are identified and discussed within the context of ASTRA and the problem elaborated in earlier chapters. The analysis should lead to a set of design choices, which can be used as a base for specifying requirements.

The *requirements specification* is based on the problem analysis. This results in requirements aimed at specifying a social tagging system in the context of ASTRA. The requirements are presented through Use Case diagrams and textual Use Cases, and finally a detailed requirements list.

The remaining contributions are made in the context of UbiCollab, and thus are contributions to the UbiCollab platform. The *design of platform architecture* proposes an architecture which aims at fulfilling the requirements, as well as integrating with existing concepts and services of the UbiCollab platform.

The *platform service APIs* contribution consists of the specification of Application Programming Interface (API)s for the platform services identified in the design part. These are the Tag Manager Client service and the Tag Manager Server service. Both services are implemented to work in the UbiCollab platform, thus constituting the *implementation of platform services* contribution. Finally, a *demonstrator tool* is made to demonstrate the platform services' functionality through the use of scenarios.

1.6 Research method

A diagram illustrating the research method used in this project is shown in Figure 1.1. The first part of the project is performed within the context of ASTRA. The work starts with a problem elaboration which discusses the problem domain and introduces important concepts, supported by related research. A scenario is also used, to further illustrate the problem domain. The next step is the problem analysis, which analyses the problem with a specific focus on identifying and discussing design issues for social tagging systems. This leads to a requirements specification, where Use Case diagrams and textual Use Cases are used and result in a detailed list of platform requirements. The problem analysis together with the requirements specification constitute the starting point for the design phase. At this stage, important UbiCollab concepts and platform architecture are introduced, as the design and implementation of the tagging system is meant to extend the UbiCollab platform. Next, an implementation of the design is created, and the functionality of the implementation is shown through a demonstration using scenarios. Finally, an evaluation of the work done in this project is performed.

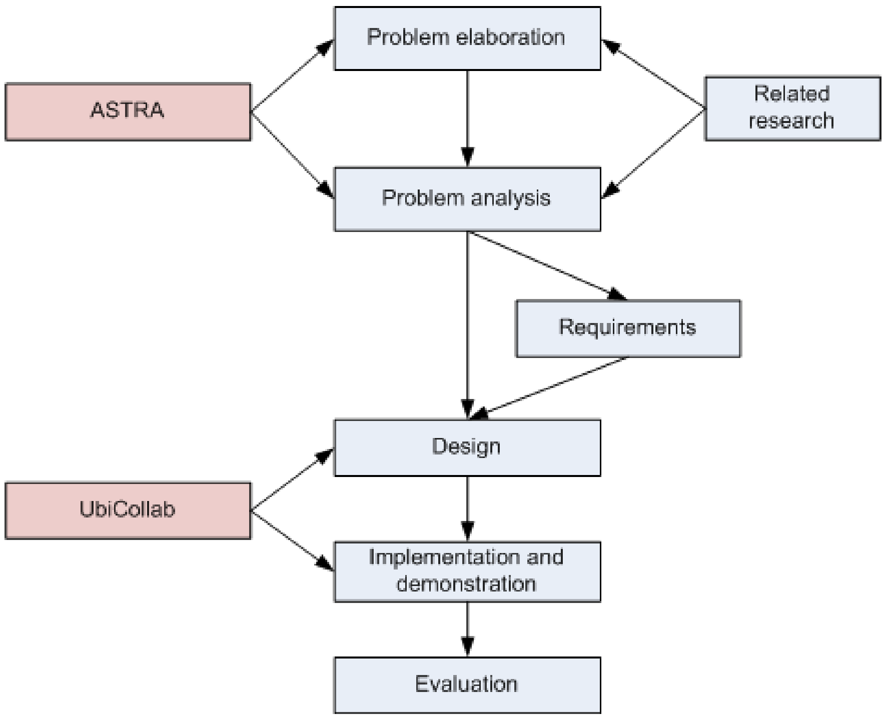


Figure 1.1: Research method

1.7 Report structure

The rest of the report is structured into the following chapters:

Problem elaboration - The problem elaboration introduces important concepts related to social tagging, and discusses these in the context of this project. A scenario is presented to further illustrate the problem and possible uses of social tagging. The chapter ends with dividing the project's main goal into subgoals.

Problem analysis - In the problem analysis, several important design issues for tagging systems are identified. Each issue is discussed in relation to the problem, and lead to a set of design choices. The second part of the chapter looks at how the tagging system can provide mechanisms for browsing and searching for services.

State-of-the-Art analysis - The State-of-the-Art analysis looks at related projects and presents a comparison of the design choices made for each of these.

Requirements specification - This chapter presents the requirements found, based on previous chapters. First, Use Case diagrams and textual Use Cases are presented. This is followed by a discussion of requirement types and a detailed requirements list.

Design - The design chapter starts by introducing UbiCollab concepts which are important for the design and implementation of a social tagging system in UbiCollab. The concepts are discussed to find how the tagging system best can be integrated with UbiCollab. This results in an overall architecture, based on a client/server solution. The components of the architecture are presented, including APIs for the platform services.

Implementation - In this chapter, the actual implementation of the platform services is presented. Also the implementation of a tool for providing access to the platform tagging functionality is presented.

Demonstration - This chapter demonstrates and evaluates the design and implementation. The demonstration is given through scenarios, showing possible uses of the tagging system and how these are supported by the implementation. The second part of the chapter evaluates the design and implementation.

Conclusions - This chapter concludes the report. A discussion of the work's contributions is given and an evaluation of the project work. Also, suggestions for further work are presented.

Chapter 2

Problem elaboration

The purpose of this chapter is to get a better understanding of the problem domain. To do this, important concepts and related research is discussed in the context of the problem domain introduced in the previous chapter. A scenario is presented and discussed to further elaborate the problem and to illustrate possible uses of social tagging. Finally, the project's main goal, presented in the introduction, is divided into a set of subgoals.

The chapter starts with a brief look at why users tend to benefit from helping each other rather than using expert information, in Section 2.1.

Section 2.2 continues with social tagging, which is a method where users help each other, collaboratively creating and sharing metadata. Social tagging is discussed in relation to ASTRA, with an emphasis on tailoring and ubiquitous computing.

Section 2.3 introduces and discusses folksonomies, which is a possible use of social tagging where the purpose is to create a folk taxonomy of the tagged items.

Section 2.4 discusses the relations between social tagging and ontologies, and why ontologies are important even when introducing a social tagging system.

In Section 2.5, a scenario is presented and discussed, to illustrate social tagging in the context of end user development in ubiquitous collaborative settings.

Finally, Section 2.6 defines a set of project subgoals.

2.1 Users helping each other

Examples of users helping each other out, sharing their experiences, are evident in many different settings. Within software development, users of a programming environment more than often share code snippets, tips and tricks with others. Among users, forums and other types of discussion tools are very common for sharing opinions and solutions to problems. Often, users ask each other for help, rather than exploring the problem on their own. For example when wanting to create a macro in a word processor, a user will often turn to friends or colleagues to borrow “their” macro, and then possibly tune it to his specific needs. One of the key advantages and reasons for why users choose this approach is the potential of saving time, not having to spend time on things that are not relevant and rather focus on their activities and interests [2]. Also, the semantic gap mentioned earlier means that fellow users are more likely to have a similar perspective to the problem domain, making it easier to understand each other than using expert information.

2.2 Social tagging

Metadata is information about data, usually meant to serve functions such as administration, structuring or description of the target data. In large collections of items, such as libraries, document repositories and photographic databases, metadata plays a vital role in organising the items, facilitating effective search techniques.

Traditionally, the creation of such metadata has been done by professionals [12]. Although this usually gives high quality metadata, there are important problems such as the cost of production and keeping up with large amounts of new content [10]. Maybe even more important is the semantic gap that can occur. An alternative to professionally created metadata is metadata created by authors, the original creators of the target data. Again there are important problems, such as inadequate or inaccurate descriptions, or outright deception [10].

A common problem for professional and author created metadata is that the eventual users of the information are not part of the creation process [10]. This leads to a third approach, focusing on users and the possibility of users collaboratively creating metadata. With user created metadata, the eventual users themselves can create the metadata, both for individual use and for sharing with each other.

Social tagging is a way of enabling user created metadata through a simple process where users attach labels, popularly called tags, to the items of interest. The metadata collected is referred to as democratic metadata, since it is generated by both creators and consumers of the content [18].

2.2.1 Social tagging and tailoring

Service composition is a high level approach to tailoring, which relies on service descriptions for users to understand and find appropriate services. A common way of describing services is to use a formal language, such as an ontology.

To describe services with a formal language, one would first define the language’s vocabulary and structure, and then define services within the language. This gives a formal, structured classification of services, especially suited for computer interpretation. However, there are important disadvantages with this approach. Maybe most important is that users most likely don’t have sufficient knowledge about formal languages and find them difficult to use. Thus, formal languages tend to be an expert approach. Also, it could be a difficult task to create a formal language for a large number of different services, which matches different people's and the system's view [19].

An alternative to using a formal language is to use social tagging, which would allow users to collaboratively describe services by tagging them. Reading a service's tags, a user would most likely get a better understanding of the service and be better able to decide if the service is appropriate for his needs. An example of tagged services is shown in Figure 2.1.

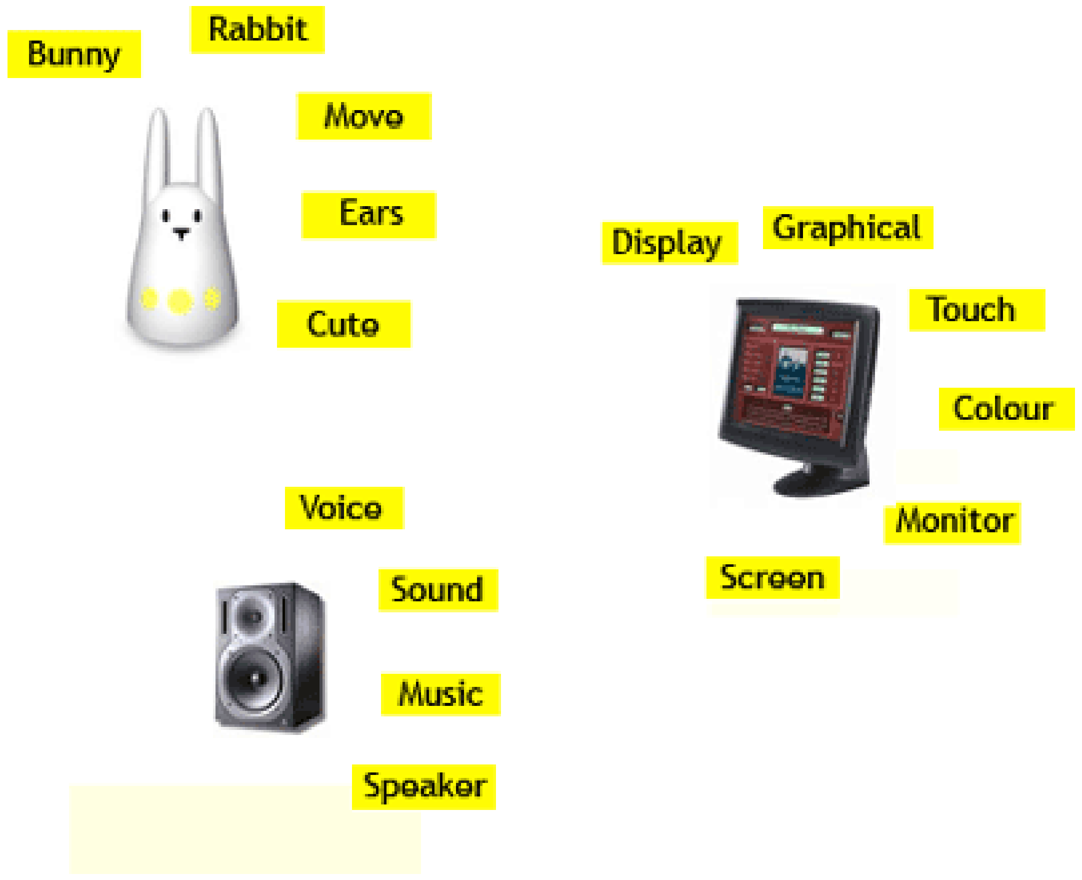


Figure 2.1: Example of services with tags

2.2.2 Social tagging in ubiquitous computing

Because ASTRA is a ubiquitous computing system, the use of social tagging would naturally be different in some ways compared to systems such as del.icio.us[1] and flickr[2]. For these systems, the objects being tagged are Internet resources (webpages or photographs), that do not reside in the physical world. Although services in ASTRA can be non-physical, they will in many cases involve some

[1]http://del.icio.us/
[2]http://www.flickr.com/

sort of physical device. For example, a service could be a touch screen monitor at the train station providing the possibility of browsing through train timetables.

Because many services will have some sort of relation to the physical world, the tags created might be affected. For example, a service's tags could be affected by its physical location. It seems natural that users would tag a service located in the library with the tag "library". Although a tag like this would not say anything about the service itself, it gives valuable information to the user. Also, tags such as this one would enable searching based on location.

As well as affecting the tags created, services' relation to the physical world also give new possibilities of interaction with the tagging system. For example, the tags describing the service could also be to some degree physical, by making them available through a Radio Frequency Identification (RFID) type of interface. This would mean that users could interact with a service by walking over to it, using a tool to read the tag interface, and then be presented with the service's tags.

2.3 Folksonomies

Social tagging provides a way of collaboratively describing items by applying tags. However, as social tagging can occur in many forms and types, it is necessary to consider the purpose of using social tagging. Folksonomies can be seen as a type or a purpose of social tagging, where the idea is to classify or categorise items.

The term folksonomy is a combination of the words "folk" and "taxonomy", and was first used by Thomas Vander Wal in a mailing list discussion [14]. Although there are debates on the definition and use of the term, there seems to be an agreement on its relation to classification or categorisation of items. For example, [6] defines folksonomy as a type of distributed classification system. In [10], a folksonomy is referred to as an organic system of organisation. Further, it is argued that folksonomies are more a way of categorisation than classification, because categorisation is generally less rigorous and boundaries are less clear.

2.3.1 Advantages

One of the most important benefits of folksonomies, and social tagging in general, is the simplicity of tagging. There is no complicated structure that needs to be learned, which means that even users with little or no competency can use the tagging system. This is one of the main motivations for introducing a tagging system, because formal languages are hard to use and learn, thus excluding the average user from the creation of metadata.

The metadata created by tagging naturally supports various searching techniques based on the tags. But in addition to traditional searching, an interesting possibility is what is referred to as tag-based navigation. Although folksonomies do not provide a formal, structured classification, there are relations that can be identified between items, tags and users. These relations can be used to provide navigation links. For example, a user can see who has tagged a given resource, and then check which other tags this person has created. In this way, it is possible to discover other users with similar interests and perspectives [18]. Since resources most often will have several tags attached to them, it is possible to compare the items' tag sets and find similar, related items. Links to related items give users a way of navigating among items, possibly leading to serendipitous discoveries.

2.3.2 Disadvantages

Although folksonomies have some clear advantages over formal languages, this does not come without a price. For example, folksonomies have been criticised because of flaws that are not inherent in formal classification systems. Examples of such flaws are homonyms and synonyms. Homonyms are words that have multiple meanings, while synonyms are words that have the same or similar meanings. Another problem is that tags can sometimes have subjective meanings. For example the tag "useful" indicates that the tagged resource is somehow useful for the person adding the tag, but the resource is not necessarily useful for other people. These problems can lead to low quality of tags. Combined with the fact that the amount of tags can become very large, the system can simply become non-navigable [18].

2.3.3 Tagging characteristics

A critical characteristic of tagging systems is that vocabularies emerge organically from the tags chosen by individual users, rather than imposing controlled vocabularies [13].

Personal tendency and community influence are important factors that most likely affect how people apply tags [13]. Users have preferences and beliefs about the tags they apply. A new user has an initial personal tendency based on previous experiences, interests and knowledge. But this personal tendency evolves as the user interacts with the tagging system. Communities influence tag selection by changing a user's personal tendency.

One might think that as a resource is tagged over a longer period of time, with new users emerging and adding their tags, the result is a chaotic pattern of tags. But research shows that the combination of many users' tags leads to a stable pattern, where the proportions of each tag used remains close to the same. Further, a stable pattern emerges at as low a number of tags as around 100 [5]. Part of the reason for this convergence is that users tend to negotiate and agree on which tags to use [19].

2.4 Ontologies

So far, social tagging and folksonomies have been discussed in relation to service descriptions. A tagging system can provide a simple method of collaboratively describing services, which is more suitable for users than formal languages. Although this has been presented as an alternative to formal language approaches to describing services, formal languages still play an important and necessary role. For service composition to work in the first place, the system must be able to decide which services can be composed together. And when services are composed together, how do they function and interact? To answer these questions, a structured formal language such as an ontology is needed.

This means that the idea of introducing a tagging system for describing services is not meant as a replacement of formal language descriptions, but rather as a complementing tool aimed at users.

2.5 Scenario

In this section, a scenario is presented together with a short discussion for each of the scenario's main actions. The idea is to relate the problem elaboration to a real scenario, showing possible uses of social tagging in the context of this work.

Action 1 *Marius has moved into a new house and he gets an electronic rabbit from Pamela. Well, it's kind of cute... but what to do with it.*

Because services differ a lot in type and functionality, understanding a service is not necessarily trivial. What can the service do and how does one use it? Services embedded in physical devices can be even worse to understand, as these introduce the extra dimension of the physical world. The user might find help in reading the service specification, but this is not very likely if he lacks the necessary computer skills.

Action 2

He reads the official information describing the service but he cannot figure out anything more innovative than using it as an advanced alarm clock.

As discussed earlier, a major problem with professional descriptions of services is that they're seldom written from the user's perspective and often miss information which is important for the average user. Services, especially physical devices, can often be used in many different ways which are hard to foresee by service creators and authors of official descriptions. This means that the official information would probably only suggest a fraction of the multitude of ways the service can be used and ways the service can be combined with other services to form applications. Thus it can be more interesting and helpful to read about other users' experiences and creative solutions.

Action 3

He then checks if anybody has tagged the service in his community. None of his closest friends seem to have used that device before or at least nobody has bothered to add a tag. So, he searches for tags left by a community of students who share a passion for innovative technology.

Providing different levels of tagging can help the user to easier find information he is looking for. Users within a community often share or have similar opinions. Thus, browsing tags in one of the user's communities can increase the quality and relevance of tags seen from the user's perspective.

Action 4

There he finds an annotation that points out that the service can be used as output device for an awareness service that he also uses with his friends. He follows the link provided in the annotation and finds out how to set up his new device to do it. Now the rabbit will blink every time that Pamela is online. Much more fun than the usual icon on his desktop.

There are many possibilities for tag types and formats used in social tagging systems, ranging from simple to complex types of tags. Video, audio, images and text are just some examples of the many possibilities to choose from. The scenario shows an example where the tag is a link pointing to a resource. The type of tag used naturally depends on the purpose of the social tagging system. If the purpose is to classify, it is natural to use simple text labels, possibly only consisting of a single word. If the purpose is to give suggestions and comments, tags consisting of several words and sentences are more appropriate.

Action 5

There's only half an hour left until the deadline of Marius's report, and he urgently needs to find a suitable printer. He's in the main building of the university, but has no clue where the nearest printer is. He brings out his PDA and searches for "printer". The results show there are two printers near him, but looking closer he sees that one of them has been given tags such as "slow" and "unstable". Obviously, he chooses the other one.

In this part of the scenario, a typical use of tags is shown. The user is provided with a searching mechanism which finds services with tags matching the search keywords. This also illustrates a different way to find services compared to finding services through service discovery mechanisms.

The scenario also suggests the use of tags for a rating system, which could also be considered as a sort of social tagging. The tags can be numbers or words indicating how good or bad the item is, thus serving an evaluative function. The Internet Movie Database[3] is an example of a popular rating system for movies, where users can rate a movie on a scale from 1 to 10.

[3]www.imdb.com

2.6 Subgoals

The main goal of this project is to design and implement a framework for social tagging of services in a ubiquitous computing context, as stated in the introduction chapter. Based on the problem elaboration, the goal has been further divided into subgoals:

- **Subgoal 1**: Identify design space of social tagging and define design choices
- **Subgoal 2**: Identify mechanisms for tag based navigation and search
- **Subgoal 3**: Define requirements for a social tagging system
- **Subgoal 4**: Propose a design solution in the form of services and tools, extending the UbiCollab platform
- **Subgoal 5**: Implement, demonstrate and evaluate a prototype of the solution

The first subgoal involves analysing the design space of social tagging of services (*subgoal 1*). By identifying design issues and discussing these in the context of ASTRA, appropriate design choices can be made. It is also important to look at how the basic social tagging functionality can provide mechanisms for tag based navigation and search (*subgoal 2*). The design choices and identification of tag based mechanisms for navigation and search form high level requirements, which can be used to define a detailed requirements specification for a social tagging system (*subgoal 3*). This is an important step for the success of designing and implementing the system. The next goal is to propose a design solution which extends the UbiCollab platform and fulfils the requirements specified earlier (*subgoal 4*). Finally, the last goal (*subgoal 5*) involves implementing a prototype of the designed solution. The implementation can then be used for demonstrating and evaluating the solution.

Chapter 3

Problem analysis

This chapter analyses and discusses important aspects of the problem which was elaborated in Chapter 2. The purpose of the discussion is to find high-level requirements for providing social tagging by considering the design space of tagging systems. The analysis of design issues is based on the elaboration, relevant research and own knowledge. The findings presented here are further used in the requirements specification in Chapter 5.

Section 3.1 starts by refining the purpose of the social tagging system.

Basic mechanisms in a social tagging system are looked into in Section 3.2.

Section 3.3 continues by looking at how functionality for searching and browsing can be provided based on the tagging system.

Finally, Section 3.4 discusses physical access to tags and how this can be provided.

3.1 Tag purpose

The problem elaboration identified several types of tags that can be part of a social tagging system, such as comments, ratings and pointers to resources. In many ways, the type of tag depends on the purpose of the tagging system. While rating tags mainly serve an evaluative function, short textual labels can be used for classification purposes.

Although all tag types discussed earlier could be relevant and worth supporting in ASTRA, the rest of this work will focus on tags for classification purposes. This choice was made to avoid the scope of the task becoming too large. However, as the process of social tagging is in many ways the same, regardless of tag type, it should be possible to extend this work to support other tag types.

3.2 Social tagging design issues

This section looks at how the basic mechanisms of a social tagging system can be provided. To do this, several important design issues have to be addressed, such as how tags can be added to a service, who is allowed to add tags and which tags are displayed to users. The following subsections are based on design space issues presented by [13] and [9], as well as own contributions. For each issue, an explanation of the issue and possible choices is given, followed by a discussion of the issue in relation to ASTRA. The choices resulting from the discussions are summarised in Subsection 3.2.8.

3.2.1 Tag sharing

This issue regards whether a tag created by one user is available to other users and if so, to whom. In other words who has access to view the tag.

Possible choices

There are three main levels of tag sharing:

- Public
- Community/Group
- Private

A private tag is strictly personal, only available to the user who created it. On the other hand, a public tag is fully shared, which means it is available to all users. These cases represent the extremities, and do not rule out the possibility of having something in between. This could be a network or community of users, or simply a group of users that the tag owner selects. Although a single tag can only be at one of these levels, the tagging system can still provide more than one level of tag sharing.

Discussion

There are several reasons why private tagging should be included in ASTRA. First, when a service is only available to the owner, it does not make sense to share tags with other users. Second, privacy should be considered. Maybe a user could be intimidated by the fact that other users will see his tags, thus preferring to keep them secret. Finally, an important point that applies to tagging systems in general is that tags can have a personal meaning, such as “me” and “toread”. Users tend to dislike seeing other people’s personal tags of this kind [13]. Although they can be very useful for the creator, they are seldom useful for other users and are more likely to be distracting.

In ASTRA, communities play a central role. This means that a natural choice is to let users limit tag sharing to one or more communities. Generally, one should consider that services will often be limited and only accessible by a group of people. In these cases it would not make much sense to share tags with users outside the group.

An important motivation for sharing tags within a community is the potential increase in quality of tags. In communities, where users tend to have similar views, the agreement on appropriate tags would probably come about easier and faster. Similarly, ambiguities should be less likely to occur because the users to some degree have a shared understanding within the community. Because of this, research such as [18] discuss the possibility of identifying communities in tagging systems based on users' tagging behaviour. However, in ASTRA communities are already present and can be taken advantage of.

The possibility of selecting individual users to form a group of users who share the tag can also be considered. But a problem in doing so is the increase in complexity and difficulty of sharing tags. It is hard to see that a group sharing feature like this would add much value that is not already provided by supporting community sharing. Thus, the minor benefits, if any, of this extra feature are outweighed by the added complexity.

In addition to the cases of limited tag sharing, it should be possible to share tags publicly. Especially when a type of service is tagged rather than a specific instance, it makes more sense to share with everyone, since the service is not connected to a specific group of people or a specific location.

This discussion has shown that ASTRA should provide all three levels of tag sharing. For the community level, the tagging system should take advantage of communities in ASTRA.

3.2.2 Tag selection

Because a tagging system could potentially have a very large number of tags, it might be necessary to limit the number of tags displayed to users. This leads to the question of which tags to select. Should the user participate somehow in the selection, or should the system handle the selection automatically?

Possible choices

If the system provides more than one level of tag sharing, tag selection could be based on these levels. One possibility is that the user is given a choice of which level tags are selected from. Another possibility is that the system separates tags for the different levels, for example showing one set of private tags and one set of public tags. If broad tagging is used (see Section 3.2.4), the tags' popularity (number of times added) can be used to select a subset representing the most popular tags.

Discussion

This design choice is important in relation to ASTRA because a major part of devices in a ubiquitous computing system have small displays and limited network connections. This means that the trivial solution of displaying a full list of tags can be too expensive, both in display size required and the amount of network data that needs to be received. Thus, ways of limiting the number of tags displayed are needed.

Two main cases should be considered. The first occurs when a user is viewing service information, or browsing/searching for services. In this case, the tags are not necessarily the main focus of the user and should come second to service information and browsing and searching functionality. For similar reasons, it may be best not to require the user to help select which tags are shown. Thus, the system should select the subset of tags itself, preferably based on the popularity of each tag rather than choosing at random.

The second case is when the user explicitly requests to view a service's tags. This means that the tags are the main focus, and the user most likely wants to have the possibility of viewing all tags. However, as the user might still want to view tags at a specific level of sharing, this option should be provided. Note that although selecting a subset of tags is not needed in this case, tag popularity can still be relevant, for example to order the list of tags.

3.2.3 Tagging rights

The system can restrict who is allowed to add tags to an item. In some systems it might be most common for users to only tag items they own themselves, while in other systems the items that are tagged are not created and owned by the users.

Possible choices

One possibility is to restrict tagging to the owner of the item, which is called *self tagging*. Naturally, this only makes sense if the owner is a user of the tagging system. Otherwise, the item would not be taggable. On the other hand, tagging can be independent of who owns the item, possibly only restricted to users who have access to the item itself. This is referred to as *free for all* tagging. A mixture of these choices is *permission based* tagging, where the owner of an item can decide who is allowed to tag the item.

Discussion

Services in ASTRA can be owned by a user or a community of users. Logically, the owner or owners of a service should be able to decide who is allowed access to it. The question is, when a user has access to the service, should this automatically allow him to tag the service?

To answer this question, it is necessary to consider that the idea of introducing a tagging system in the first place was to help users collaboratively describe services. Once tagging is limited to the service owner, it is no longer a collaborative process, thus limiting the value of the tagging system. One of the reasons why a service owner might want such a restriction, is that he wishes to tag his service without other users interfering. But in that case, he can simply use the private tagging functionality. For a service owned by a community, it is possible to use community tags, which are not accessible by users outside the community.

3.2.4 Tag scope

The scope of a tag describes if the tag belongs to individual users or is shared by a community. The scope affects the aggregation of tags around items, by deciding how the system reponds to the same tag being added to an item by more than one user.

Possible choices

There are two possible choices for tag scope, called *broad tagging* and *narrow tagging* [13]. Broad tagging means that each user can add his own tags, with each tag being treated as unique. This implies that each tag has a connection to the user who created it, and in a sense belongs to the user. Narrow tagging however means that users share a common set of tags for each item, which means that tags have no connection to users. Usually tags in narrow tagging systems are created by one or a few persons.

Discussion

It is again important to point out that the goal of the tagging system is to provide mechanisms for collaboratively describing services. When only one person or a few persons provide the tags for an item, it can hardly be called a collaborative process. This suggests that broad tagging is the better choice. Another reason is that broad tagging enables additional tag based navigation possibilities (see Section 3.3), due to the connection between tags and users. This is important, as one of the aims of the tagging system is to help users find services more easily. Finally, for the purpose of classification, the number of times a tag has been added is an important indicator of the importance and relevance of the tag.

3.2.5 Tag support

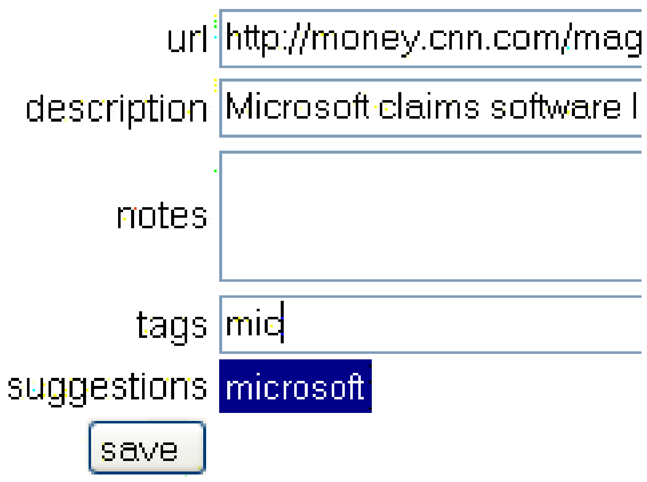
This issue regards whether the user is given help or suggestions when adding tags.

Possible choices

There are three levels of tag support. *Blind tagging* is when a user cannot see tags entered by other users while tagging. With *viewable tagging*, the user can see tags added to an item by other users while tagging. The last category is *suggestive tagging*, where the system suggests appropriate tags to the user. An example of suggestive tagging is shown in Figure 3.1.

Discussion

Research has shown that existing tags affect future tagging behaviour, thus making it possible to steer users towards creating tags of the preferred type [13]. This can be done by seeding the system or by giving suggestions of tags of the preferred type. When the purpose of the tagging system is to classify or categorise,



url http://money.cnn.com/mag

description Microsoft claims software l

notes

tags mic

suggestions microsoft

Figure 3.1: Tag suggestions in del.icio.us

these measures can be used to help improve the quality of tags, and get a quicker convergence towards a folksonomy [5]. In the case of service composition in ASTRA, it is already clear that ontologies describing services are needed. These ontologies can also be used as a base for providing tag suggestions.

3.2.6 Tag format

One of the questions that has to be answered for any tagging system is which restrictions that should be placed on the format of tags allowed. Format here means what type of characters the tag contains, and if the tag contains one or more words or even one or more sentences.

Possible choices Giving a list of possible tag formats is impossible, because an endless list of formats can be made simply by making small changes to the allowed character set. However, it is possible to give some general categories, focusing on the degree of restriction.

A liberal approach is to let users compose tags in any way they want, placing no constraints on the number of words or the type of characters allowed. This means that tags can be full sentences, such as “a very nice but slow device”. In contrast to this approach, tags can be limited to single words and only alphabetical characters.

Discussion

Because the goal of the tagging system is to classify or categorise services, tags of any length and any number of words are not suitable. If tags take on the form of comments, it is harder or even impossible to identify relations between services’ tags. This also reduces the possibility of tag based navigation. Thus, it is clear that a more restricted tag format is needed. There is however no clear indication of exactly what format is the best for classification. Most other tagging systems do not allow spaces and limit the tag length, thus limiting tags to a single word or a few words seperated by a "-". The same approach will be used here.

For private tags however, it is not necessary to place such a restriction on tag names. Private tags are only visible to the creator and are not part of the folksonomy which the tagging system aims to create.

3.2.7 Generic or specific tagging

Looking closer at what a service is, it is seen that there are two possibilities for how services can be tagged. First, one specific service can be tagged, for example a public monitor in the library. Secondly, it is possible to tag a type of service, rather than one specific instance. For example the type of monitor used in the library, which could also be in the office, canteen, and so on. This means that to implement a tagging system, it is necessary to make a distinction between the two cases.

Possible choices

The choice that has to be made in this category is whether to support tagging instances or types, or both. An illustration of the difference between tagging specific services and generic services is given in Figures 3.2 and 3.3. Both figures show a generic Nabaztag[1], which represents the type of service rather than a specific instance. In addition there are three instances, which are specific services of type Nabaztag. In Figure 3.2, only the generic service is tagged, while in Figure 3.3 each instance has its own tags.

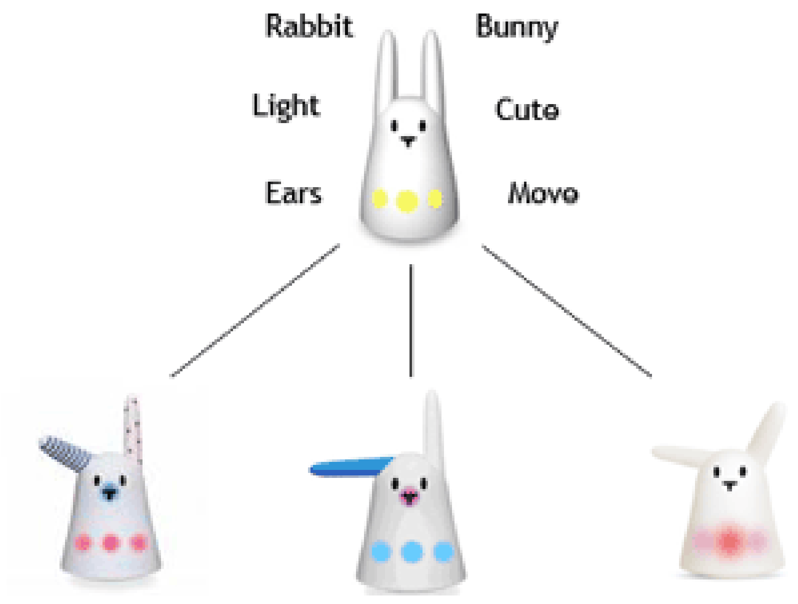


Figure 3.2: Tagging generic services

Discussion

[1]http://www.nabaztag.com

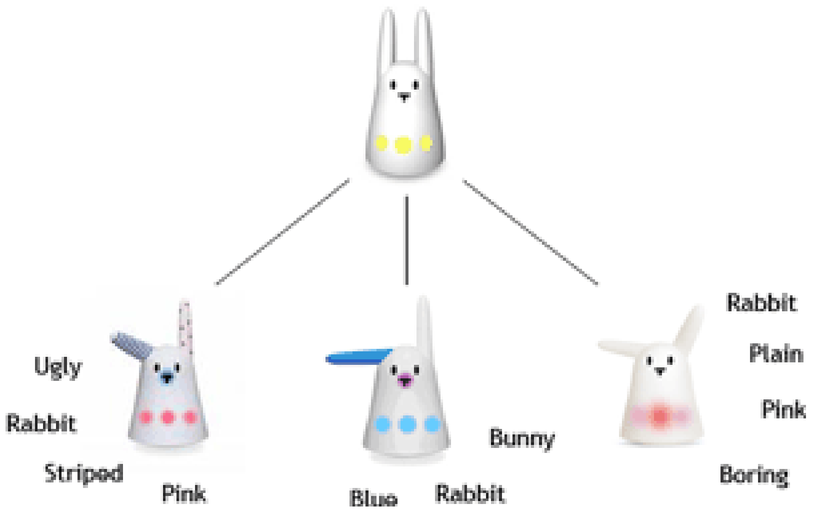


Figure 3.3: Tagging specific services

One of the main factors that could influence the choice between generic and specific tagging is whether instances of a service type differ a lot. In the case that service instances only have minor and not very important differences, using generic tagging is most likely sufficient and the simplest approach. Tagging each instance would only lead to several almost equal tag sets. But in a ubiquitous computing environment like ASTRA, service instances of the same type are more likely to differ. This is especially due to the fact that services can be embedded in physical devices, giving a multitude of possibilities for service behaviour and functionality. This means that generic tagging would only cover parts of the tags that are relevant for each instance. Important tags could be missing, and users might have problems finding an appropriate service instance.

This does not mean that generic tags are not of any value. For example, generic tags could be used to create a classification of service types. One approach would be to deduce the service type tag set from the tag sets of service instances of this type. However, this is outside the scope of this work, and should be considered in future work.

3.2.8 Summary

The design choices made are listed in Table 3.1. These can be seen as high level requirements for the proposed solution. Each design issue is presented with possible choices and the choice made.

Design issue	**Possible choices**	**Choice**
Tag sharing	Public, community, private	All levels should be provided

Continues next page

Design issue	**Possible choices**	**Choice**
Tag selection	All, limited	Both should be possible. Tag retrieval could be limited by tag level and/or tag popularity
Tagging rights	Self tagging, permission based, free for all	Free for all
Tag scope	Broad, narrow	Broad tagging system
Tag support	Blind, viewable, suggestive	Suggestive tagging based on underlying service ontologies
Tag format	Open, Semi-restricted, restricted	Restricted
Generic or specific	Type, instance, both	Instance

Table 3.1: Summary of design choices

3.3 Finding services

The previous section looked at how the basic tagging mechanisms can be provided and important design issues regarding these. Once the mechanisms are in place, users can collaboratively describe the multitude of services in ASTRA. The next step is to look at what functionality could be provided to take advantage of the tags created through the tagging system.

The idea and motivation behind introducing a tagging system for services was to help users understand and find appropriate services. In a system like ASTRA there will be a very high, constantly growing, number of services of different types. Because of this, users need to be supported by convenient and easy-to-use mechanisms for finding services.

The following subsections discuss searching and browsing for services, and physical access to services. Note that although searching and browsing are discussed seperately, they are closely related. For example, when a user is viewing a search result, browsing options such as viewing other users' tags or finding similar services can be displayed.

3.3.1 Searching

One of the most powerful and important functions enabled by the tagging system, and metadata in general, is the possibility to search. By simply entering a keyword or a list of keywords, the user can easily search for services with tags matching one or more of the keywords.

It is also possible to extend and combine searching based on tags with other attributes, such as service location. By specifying a location, the user can search for services in the given location, with tags matching the search string.

3.3.2 Browsing

What is meant by browsing services is similar to browsing webpages on the Internet. The user wants to find a service matching his requirements or simply one that looks interesting and worth trying. Although browsing is similar to searching, there is an important difference. When searching, the user explicitly asks the system to find all services matching his search parameters or query. Browsing however is not based on a formulated query. Instead, the user explores the service space by following relations between services, finding relevant information.

To enable browsing, relationships between services need to be identified. Can tags help provide these relationships? A folksonomy is not a structured formal language like an ontology. Still there are certain relationships that can be identified between the parts of the folksonomy. [9] illustrates possible relationships with Figure 3.4. First, users can add tags to items, which means there are connections between users and items. Second, users can also be connected in some way, for example by being members of the same community. A last possibility is that items are somehow related.

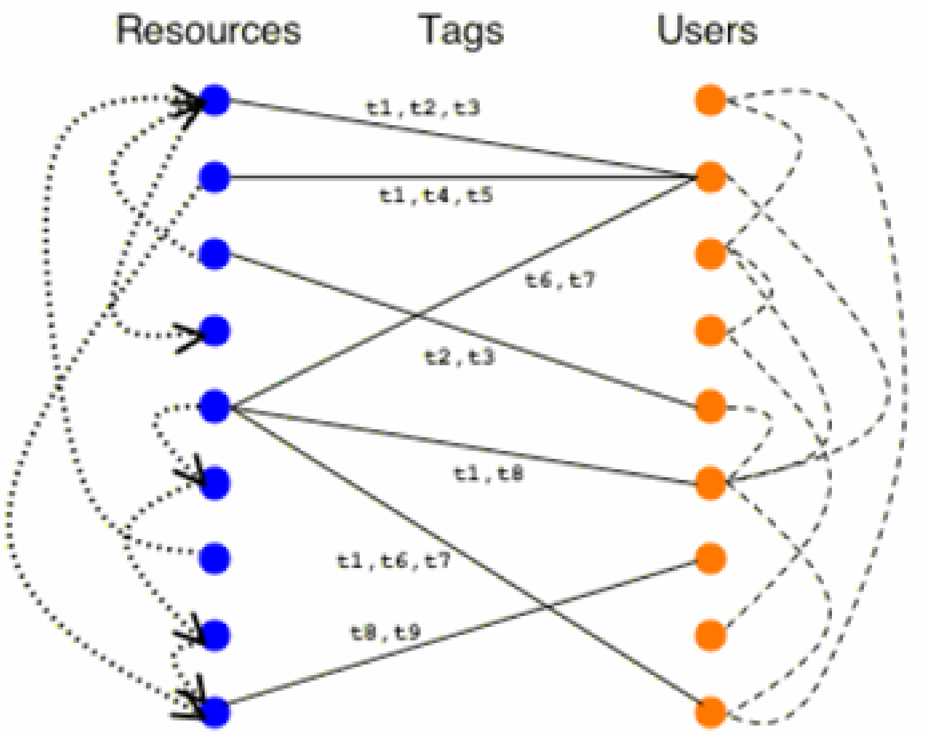


Figure 3.4: Relationships

For ASTRA, the items being tagged are services, which means there are relations between services and tags. Next, due to the importance of communities in ASTRA, a possible connection between users is that they are in the same community. Services can also be related, for example by one using information from the other, or services being in the same location. The relations identified are illustrated with an Entity Relationship (ER) diagram in Figure 3.5. With these relations identified, the next step is to look at how they can be used for browsing services.

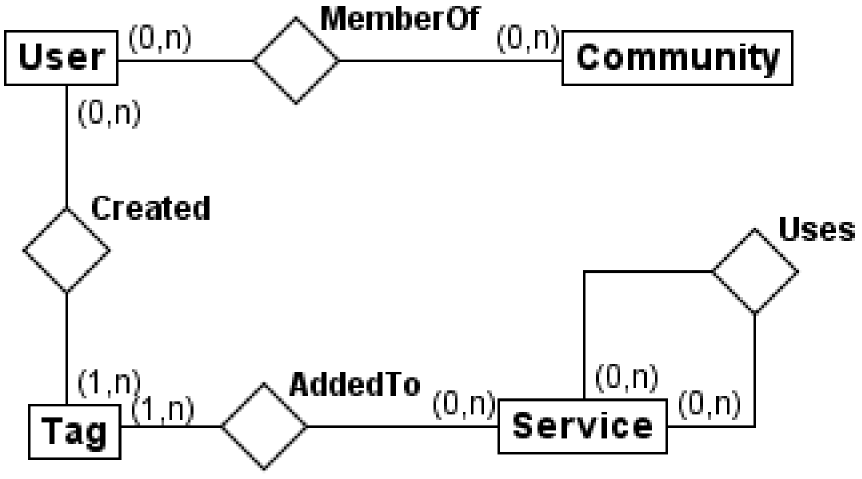


Figure 3.5: Relationships in ASTRA

View user

When viewing a service and the accompanying tags, one option is to allow viewing the user that has added a certain tag. Information, such as which other tags that user has added and which other services the user has tagged, can be given. The idea here is not to expose the creators of tags, but rather to give users additional navigation possibilities. For example, it would make it possible to find users with similar perspectives and interests [18], and also find services that these users have used. Generally these additional navigation possibilities allow casual browsing and could lead to serendipitous discoveries [18].

Related services

Although tags do not describe a service in a structured, formal way, the tagging system can still use the tags to suggest the similarity of services. By comparing services' tag sets, the system can find the services that best match, and thus present links to related services. Depending on the choice of the user, the comparison can be done at public, community, or private level. For example, the user's private tags for a service can be compared with services' public tags.

3.4 Physical access

Search and browse features of the tagging system help users find services by specifying a query or by navigating with links between services, tags and users. An alternative use of a tagging system is when the service has already been found by the user. When the user finds a service, it is likely that he is interested in knowing more about the service, what functionality it has and what it can be used for.

In a ubiquitous computing system, services are often physical devices. This means that they can be physically discovered by users or automatically discovered by the system when the user is nearby. When a user finds a service this way, the system should be able to recognise the service and present the service's tags to the user, thus giving the user a kind of physical access to service information and tags. A similar approach is used in the *Tokyo Ubiquitous Network Project* [4], where collaboratively created tags help people find their way around Tokyo. Users are given physical access to the tags through a simple pocket device. This means information, videos, pictures and other types of content regarding a specific location can be read using the pocket device.

For services, physical access to service tags can be given by integrating the tagging functionality with service discovery mechanisms of the system. Examples of such mechanisms are RFID readers and discovery protocols such as Universal Plug and Play (UPnP). Once a service has been recognised through service discovery, the tagging system can provide functionality for reading and creating tags.

Chapter 4

State-of-the-Art analysis

This chapter looks into state-of-the-art within social tagging systems. The idea of this analysis is to get a better understanding of tagging systems in general, and to see how specific tagging systems have solved common design issues.

The tagging systems del.icious, Flickr and Youtube are presented in sections 4.1, 4.2 and 4.3 respectively.

Section 4.4 presents the comparison of the tagging systems together with a short discussion of important differences.

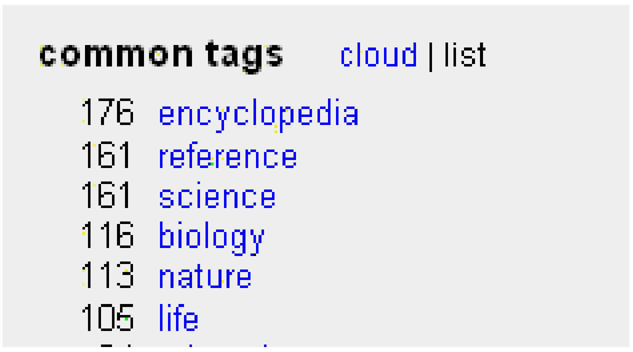
4.1 del.icio.us

del.icio.us[1] is a social bookmarking web service which was launched in 2003. Each user can create bookmarks for webpages, as well as search and browse through bookmarks created by other users. What makes del.icio.us interesting in relation to this work, is the use of a tagging system. Not only can users add descriptions to their bookmarks, they can also add tags.

Users can create private bookmarks not available to other users, but the main emphasis of del.icio.us is to publicly share bookmarks as well as their accompanying tags.

For a large number of webpages bookmarked in del.icio.us, the webpage has been bookmarked by more than one user. In these cases, the system combines each user's tags into a list of common tags. By presenting the number of times a tag has been used compared to other tags, the popularity and relevance of the tag is indicated. As well as the straightforward list of tags shown in Figure 4.1, del.icio.us also has the option of displaying tags as a tag cloud. Here, the number of times a tag has been added is indicated by the font size and weight. An example tag cloud is shown in Figure 4.2. Note however that Flickr was the first website to implement tag clouds [17].

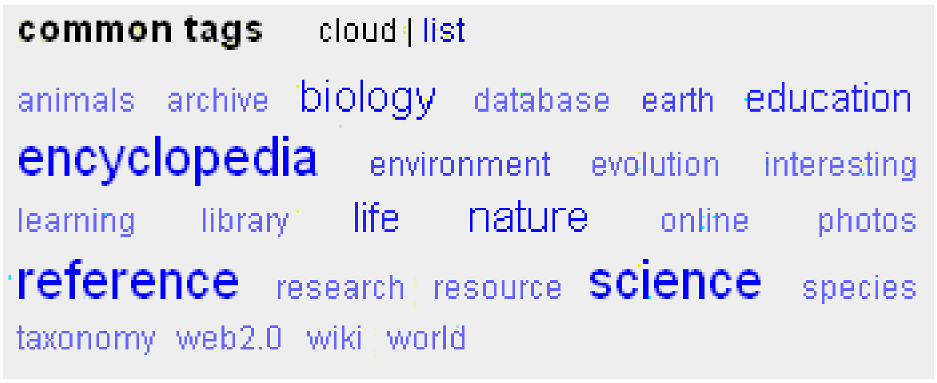
[1]http://del.icio.us/



common tags cloud | list

176	encyclopedia
161	reference
161	science
116	biology
113	nature
105	life

Figure 4.1: Tag list



common tags cloud | list

animals archive biology database earth education encyclopedia environment evolution interesting learning library life nature online photos reference research resource science species taxonomy web2.0 wiki world

Figure 4.2: Tag cloud

4.2 Flickr

Flickr[2] is a photo sharing web service, launched in 2004. A fundamental difference between Flickr and other social tagging systems is that tag creation is limited to the owner of an item. Although tags added to a public photo can be viewed by everyone as well as being searchable, it is only the user who uploaded the photo who can create and edit these tags.

Flickr provides multiple levels of access control to photos. First, a photo can be given private or public access. Private means that only the owner can view the photo, while public allows everyone access. There is also the option of making a photo part of a group. In this case, if the group is private then the photo can only be viewed by members of the group. If the group is public, the photo can be viewed by everyone.

[2]http://www.flickr.com/

4.3 YouTube

YouTube[3] is a video sharing website which has had an extreme increase in use and popularity since it was launched in 2005. YouTube is an example of a system which does not put much emphasis on the tagging features, but rather uses tags to complement descriptions, comments, ratings and video categories.

Similar to Flickr, YouTube videos can only be tagged by the users who upload them. Because of this, the video's tags do not necessarily represent users' general understanding of how the video is best described. Also, the number of tags per video is usually rather limited. This means there is no need for the system to apply a tag selection algorithm to show a subset of the video's tags.

Several mechanisms for tag-based navigation and search are provided. When finding an interesting tag, one can click the tag to search for other videos with the same tag. When a video is viewed, a list of other videos with similar tags are shown, meaning that users can easily navigate from one video to another.

Although YouTube does not have a strong emphasis on communities, certain features are provided to help users connect. For example, a user can subscribe to another user, which means that when the second user uploads a new video, the first user is notified.

4.4 Comparison

This section compares the design choices made for each of the presented tagging systems. The comparison is shown in Table 4.1.

Design issue	**del.icio.us**	**Flickr**	**YouTube**
Type of item	Webpages	Photos	Videos
Tag sharing	Public	Public	Public
Tag selection	All Limited by popularity Private	All	All
Tagging rights	Free for all	Permission based	Self tagging
Tag scope	Broad	Narrow	Narrow
Tag support	Suggestive	Blind	Blind
Tag format	Semi-restricted	Open	Semi-restricted
Generic or specific	Instance	Instance	Instance

Table 4.1: Comparison of design choices

[3]http://www.youtube.com/

One important observation regarding all these systems is that private and community tag sharing is not provided. Instead of separating into several levels, the tag sharing only relies on the item which is tagged. This means that the item's tags are publicly available to all users that have access to the item. All in all, the concept of communities is hardly used in any of the systems and only minor features for connecting to other users are provided. This means that the potential increase in quality of tags within communities, suggested by research [13], is not taken advantage of.

The difference in degree of collaboration in the systems should be noted. Both Flickr and Youtube are more in the direction of users tagging their own items, rather than users collaboratively tagging each other's items. This can be seen from the design choices "tagging rights" and "tagging scope". del.icio.us on the other hand uses broad and free-for-all tagging, which means each user can give his opinion by creating his own tags to an item. This is more of a collaborative approach, and is more suitable if the aim is to create a folksonomy [16].

Chapter 5

Requirements specification

This chapter defines and formulates requirements, based on the findings of the problem analysis. The problem analysis started by discussing how basic tagging functionality can be provided and the design issues this involves. Next it looked at how search and browse functionality can make use of the basic tagging features. This chapter uses the same separation to structure the functional requirements.

First the functional requirements of the system are presented with Use Case diagrams in Section 5.1 and further detailed with textual Use Cases in Section 5.2. Section 5.3 discusses requirement types, and presents a detailed requirements specification.

5.1 Use Case diagrams

From the problem analysis, three main categories of functionality can be identifed; Management, Browsing and Searching. This leads to the top level Use Case diagram shown in Figure 5.1.

The **Management** category contains the basic tagging functionality identified in the problem analysis. The Use Case diagram for **Management** is shown in Figure 5.2.

The **Browsing** category is responsible for functionality related to browsing services based on tags, also called tag based navigation. The Use Case diagram for **Browsing** is shown in Figure 5.3.

Finally, the **Searching** category contains functionality for searching for services based on tags. The Use Case diagram for **Searching** is shown in Figure 5.4.

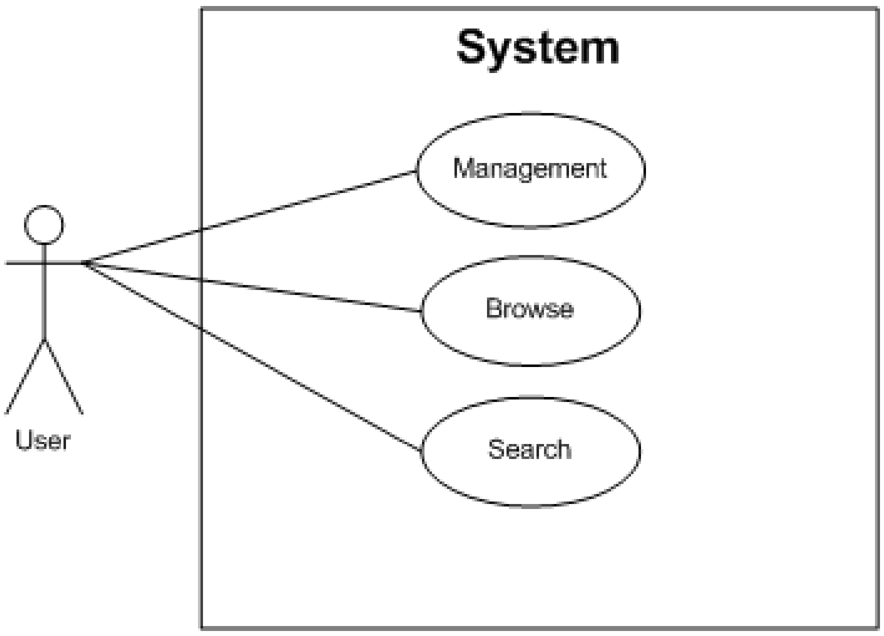


Figure 5.1: Top level Use Case diagram

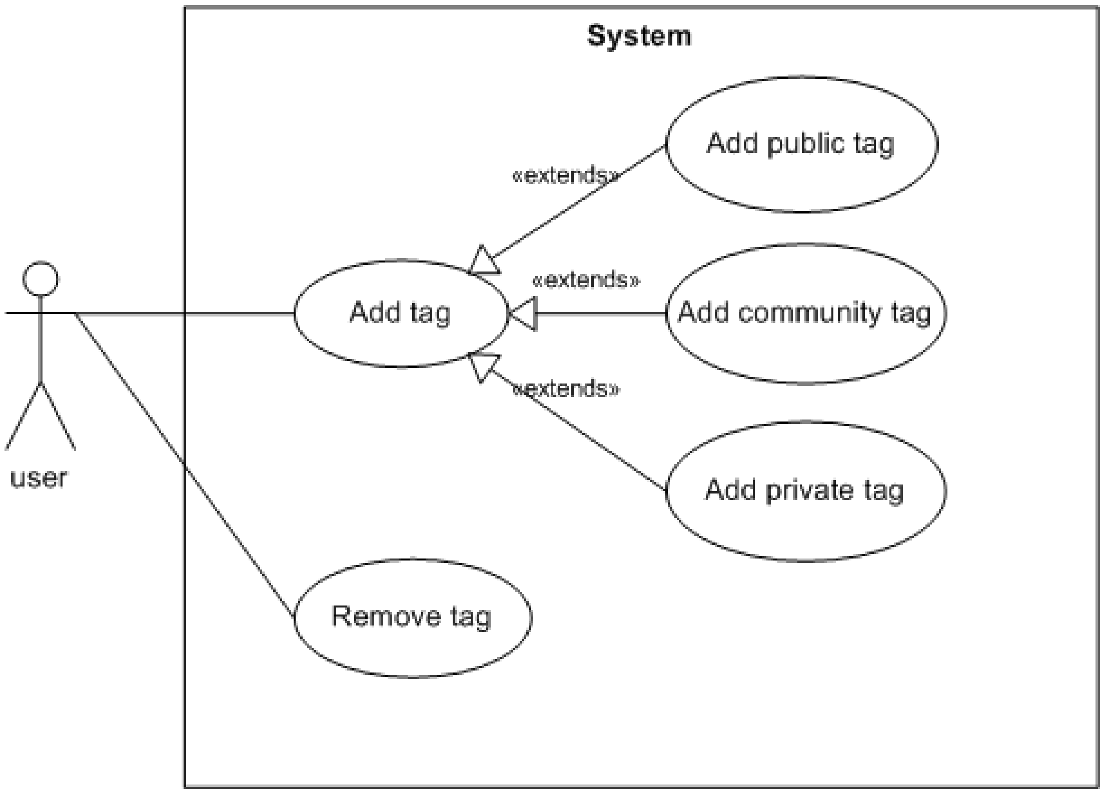


Figure 5.2: Management Use Case diagram

5.2 Textual Use Cases

This section presents the textual Use Cases. Each textual Use Case has a Main Success Scenario, abbreviated "MSS", which shows the main path of the Use

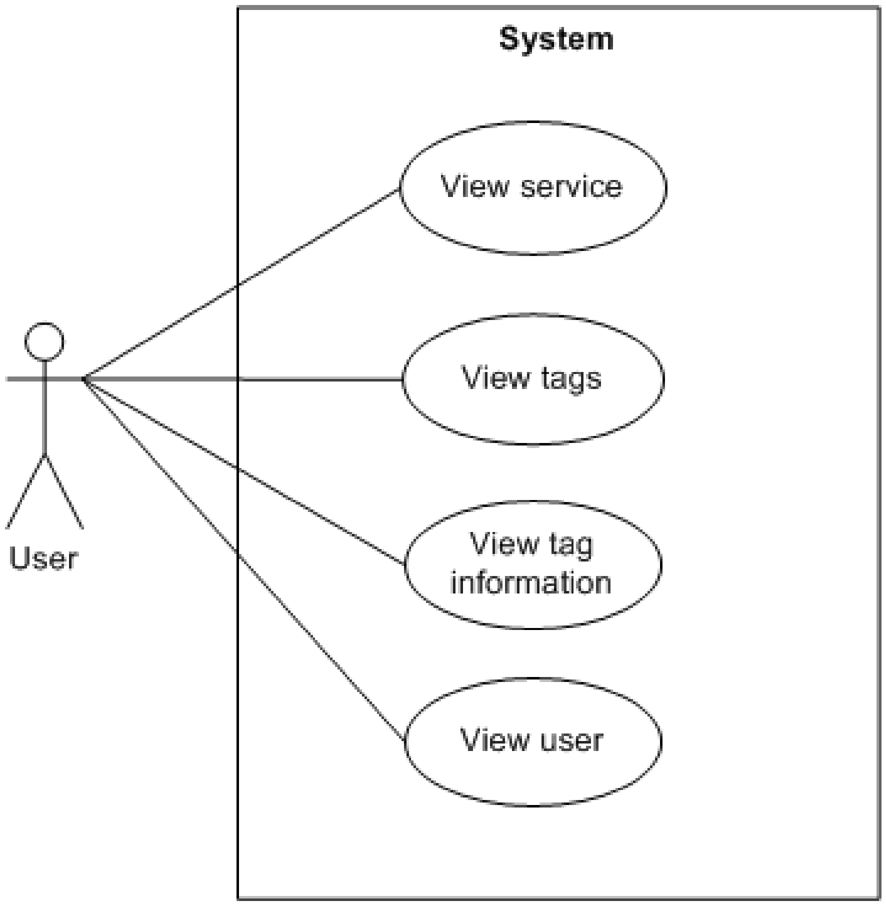


Figure 5.3: Browsing Use Case diagram

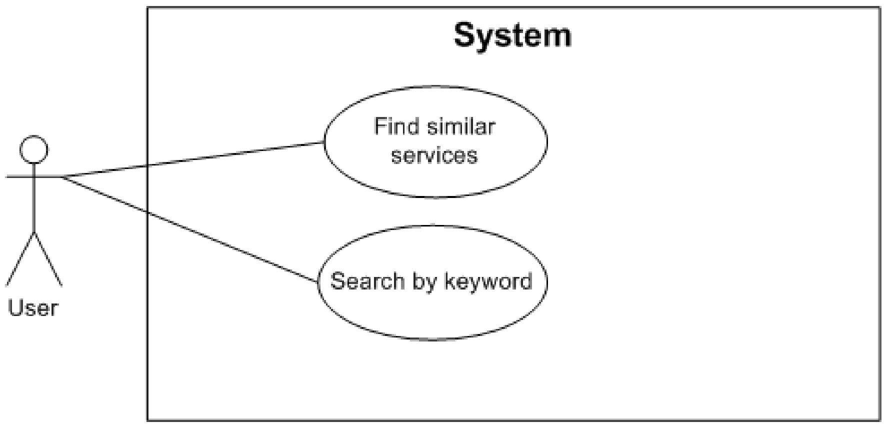


Figure 5.4: Searching Use Case diagram

Case. In addition, there can be alternativate paths, given under "Extensions". Underlined text is a reference to another Use Case.

5.2.1 Management

This subsection contains textual Use Cases for the **Management** category. Table 5.1 presents the Use Case *Add tag* and Table 5.2 presents the Use Case *Remove tag*.

UC1	Add tag
Description	The user adds a tag to a service
Primary actor	User
Preconditions	User has found a service
MSS	1 User selects tag visibility 2 User enters a tag name 3 System validates the tag name 4 System confirms that the tag has been added
Extensions	4a Tag is not valid .1 System tells the user why the tag is not valid, return to MSS at step 3

Table 5.1: Use Case 1: Add tag

UC2	Remove tag
Description	The user removes a tag he has previously created
Primary actor	User
Preconditions	User has previously added a tag to a service
MSS	1 User selects a tag 2 System displays option for removing tag 3 User chooses to remove the tag 4 System confirms that the tag has been removed
Extensions	2a Selected tag was not created by the user .1 System does not display option for removing tag

Table 5.2: Use Case 2: Remove tag

5.2.2 Browsing

This subsection contains textual Use Cases for the **Browsing** category. Table 5.3 presents the textual Use Case for *View service*. Table 5.4 presents the textual Use Case for *View tags*. Table 5.5 presents the textual Use Case for *View tag information*. Finally, Table 5.6 presents the textual Use Case for *View user*.

UC3	View service
Description	The user views a service
Primary actor	User
Preconditions	
MSS	1 User selects a service to view 2 System displays service information and a list of the most popular tags
Extensions	

Table 5.3: Use Case 3: View service

UC4	View tags
Description	The user is viewing a service and wants to view a complete list of the service's tags
Primary actor	User
Preconditions	User is viewing a service
MSS	1 User chooses to view the service's tags 2 System finds all the service's public tags 3 System displays a list of the tags found, with the number of times each tag has been added
Extensions	2a User wants to view only private tags .1 System finds all the service's tags created by the user, return to MSS at step 3 2b User wants to view community tags .1 System displays a list of communities .2 User selects a community .3 System finds all the service's tags that have been created by members of the selected community, return to MSS at step 3
	Continues next page

UC4	View tags

Table 5.4: Use Case 4: View tags

UC5	View tag information
Description	The user is viewing tags of a service, and wants to view more detailed information of a specific tag
Primary actor	User
Preconditions	User is viewing a service's tags
MSS	1 User selects a tag 2 System displays a list of users that have added this tag 3 System displays a list of other services with the same tag
Extensions	

Table 5.5: Use Case 5: View tag information

UC6	View user
Description	The user is viewing a tag of a service, and wants to view more detailed information of one of the users that have added the tag
Primary actor	User
Preconditions	User is viewing a tag
MSS	1 User selects a user that has added the tag 2 System displays a list of other tags the selected user has added 3 System displays a list of other services the selected user has tagged
Extensions	

Table 5.6: Use Case 6: View user

5.2.3 Searching

This subsection contains textual Use Cases for the **Searching** category. Table 5.7 presents the textual Use Case for *Search by keywords* and Table 5.8 presents the textual Use Case for *Find similar services*.

UC7	Search by keywords
Description	The user searches for services with tags matching the given keywords
Primary actor	User
Preconditions	
MSS	1 User enters one or more keywords 2 System compares services' tag sets with the keywords to find matching services 3 System displays services that matched
Extensions	

Table 5.7: Use Case 7: Search by keywords

UC8	Find similar services
Description	The user wants to find services that are similar to the selected service
Primary actor	User
Preconditions	User has selected a service
MSS	1 User chooses to find similar services 2 System finds similar services by comparing the services' tag sets with the selected service's tag set 3 System displays the resulting services
Extensions	

Table 5.8: Use Case 8: Find similar services

5.3 Functional requirements

This section formulates and presents the functional requirements found through problem analysis and Use Cases. The aim is to come closer to a system specification, which can be used as a base for design and implementation. The section starts with an explanation of different requirement types, in Section 5.3.1. Then requirements for basic tagging functionality are presented in Subsection 5.3.2 and requirements for search and browse functionality are presented in Subsection 5.3.3.

5.3.1 Requirement types

When analysing requirements for ASTRA, it is important to decide and clarify what type of requirements are sought. Requirements can be categorised based on which part of the system they apply to. One of the main challenges is to make the platform provide sufficient functionality to serve as intended, without it growing out of proportions. Too little functionality, and the platform is pretty much useless. Too much, and the platform could end up being too complex and hard to use. In other words, requirements that are only relevant in a few settings most likely belong to specific applications built on top of the platform.

Based on the aim of ASTRA to develop a platform with accompanying end user tools, requirement types can be divided into three categories:

Platform

The platform is the core of ASTRA, which contains the base functionality needed for ASTRA applications. Essential and often used functionality should be part of the platform.

End user tools

End user tools are software mechanisms that will help end users create awareness applications [1]. Thus, requirements in this category are requirements for these software mechanisms.

Applications

This category consists of requirements for specific applications, built to function on top of the platform.

This project focuses on the platform and accompanying end user tools, rather than specific applications. This means that the requirements sought are not limited to one application or one possible use of ASTRA, but general requirements needed for ASTRA to serve as a base for building a multitude of applications.

5.3.2 Basic tagging functionality

This subsection presents the requirements for basic tagging functionality. The requirements are listed in Table 5.9.

ID	Description
FA1	It should be possible to create new tags for service instances
FA2	Tag name suggestions should be given based on the underlying service ontologies
FA3	A tag should be possible to remove by the user who created it

Continues next page

ID	Description
FA4	A tag's visibility level should be possible to set when creating it
FA4.1	It should be possible to set the tag as private
FA4.2	It should be possible to set the tag as public
FA4.3	It should be possible to set the tag as visible to one or more communities
FA5	Different users should be allowed to create the same tag to the same service. The tags should be treated as unique
FA6	A user should be prevented from creating the same tag more than once for a service, within the same visibility level. For communities, this means once per community
FA7	Public and community tags should not be allowed to contain spaces and should not exceed a maxium length
FA8	It should be possible to query the system for a service's tags
FA8.1	It should be possible to retrieve tags at level public, private or community
FA8.2	It should be possible to retrieve only a part of the resulting tag set
FA8.3	It should be possible to retrieve only the most popular tags in the resulting tag set
FA8.4	It should be possible to find the number of times each tag name has been added at public or community level
FA9	The platform must provide a suitable, platform independent, interface to the platform functionality

Table 5.9: Basic tagging requirements

5.3.3 Browse and search functionality

This subsection presents the requirements for search and browse functionality. The requirements are listed in Table 5.10.

ID	Description
FB1	It should be possible to query the system for the users who have tagged a service with a given tag
FB2	It should be possible to query the system for other services tagged by a given user
FB3	It should be possible to find who has added a tag
FB4	It should be possible to find which other tags a user has added

Continues next page

ID	**Description**
FB5	It should be possible to find which other services a user has tagged
FB6	It should be possible to retrieve services that have been tagged with a given tag
FB7	It should be possible to search for services by keywords.
FB8	It should be possible to find services that are similar to a given service, based on the services' tag sets.

Table 5.10: Browse and search requirements

Chapter 6

Design

This chapter proposes a design solution for the tagging system. The aim of the design is to fulfil the requirements presented in the previous chapter, within the framework of UbiCollab. This should lead to an overall tagging system architecture.

Section 6.1 introduces relevant UbiCollab concepts.

Next, Section 6.2 discusses how the tagging system can extend UbiCollab.

Section 6.3 discusses the representation of tags.

Section 6.4 looks at deployment issues.

Finally, the architecture resulting from the discussions is presented in Section 6.5

6.1 UbiCollab concepts

This section discusses important UbiCollab concepts, as presented by the UbiCollab whitepaper [3]. The concepts are essential for the design and integration of a tagging system into UbiCollab.

6.1.1 Collaboration spaces and instances

Collaboration Space (CS) and Collaboration Instance (CI) are important terms used by the UbiCollab architecture [3]. A CS is a physical location that could contain a number of users and physical devices. Examples are the office and the home. CIs are virtual contexts for collaboration among users from one or more CSs. A CI could contain information such as participant details and collaboration history. Simply put, the CI is where users communciate and share information.

These concepts are illustrated in what UbiCollab calls the human grid, shown in Figure 6.1. Physical locations are represented by CSs, each with one or more users. The CI in the middle is the virtual context for collaboration between the users.

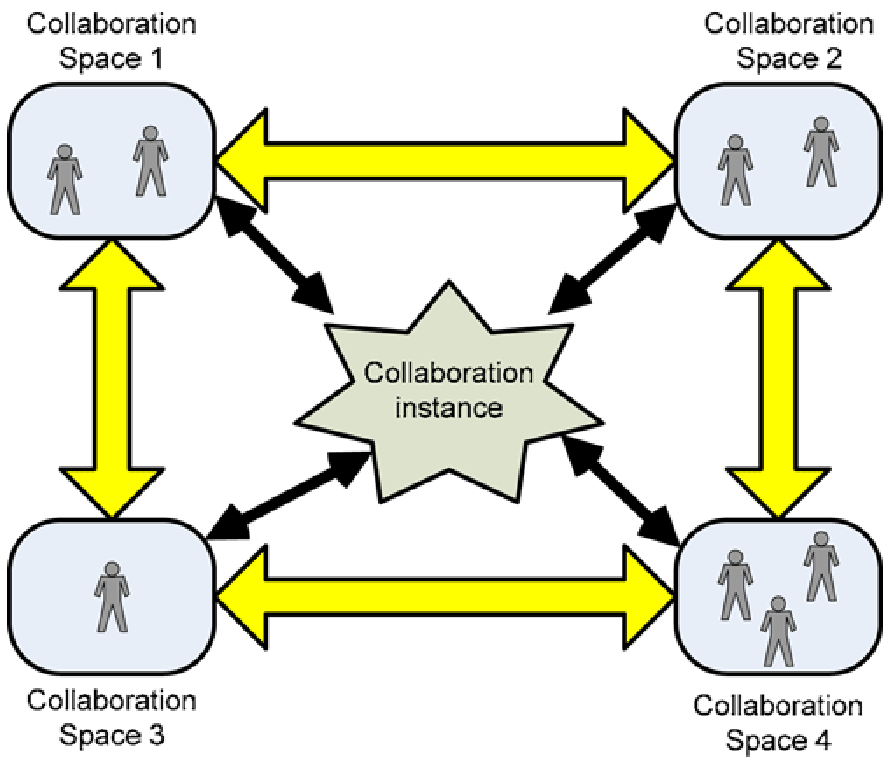


Figure 6.1: Human grid

In many ways, the CI is similar to a community. When users participate in a CI, they are part of a group of people, most likely with common interests. This could be a group of students working on the same project, company colleagues, or family members.

6.1.2 Services

Services in UbiCollab are external devices, artifacts or web services, represented by Service Proxy (SP)s. A SP is an API that describes how to interact with the service, and can be seen as a sort of service driver.

One of the main ideas in UbiCollab is that services can be shared with other users through publishing them to CIs. For this to happen, first the service has to be discovered using a discovery protocol such as RFID or Universal Description, Discovery and Integration (UDDI). Next a SP representing the service can be installed in the user's Service Domain (SD). At this point the service, as well as other services in the user's SD, can be published to CIs that the user is participating in. Figure 6.2 illustrates the concept of a SD.

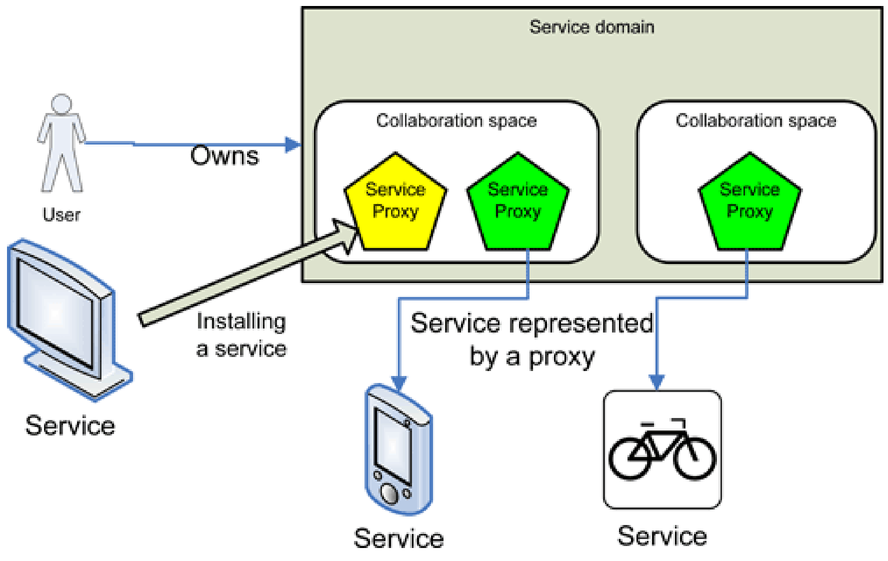


Figure 6.2: Service domain

6.1.3 Service Registry

The Service Registry (SR) is a repository of services. The idea in UbiCollab is to have one SR for each UbiNetwork. In other words one SR acts as a repository of all services within a UbiNetwork. Each service has several attributes which are stored as service details in the SR. Among these are name, owner and location. In addition each service has two pointers, the description Uniform Resource Identifier (URI) and the service URI. The description URI is a pointer to the service's description. The service URI is a pointer to the invocation point of the service, or in other words the service's address. As this address is unique, it can be used as an identifier for the service.

There is no tight coupling between SR and SD, however there are a few relations. First, it is possible to find services in the SR which can be installed in the SD. Further, it is possible to "advertise" your services (installed in SD) in the SR.

6.1.3.1 Service Discovery Manager

The Service Discovery Manager (SDM) service, which runs locally on a user's UbiNode, handles discovery of services. It supports different discovery protocols, such as RFID, Bluetooth and UPnP. It is possible to register as a client to SDM, "listening " to service discoveries. This means that whenever SDM discovers a service, clients are notified. SDM also supports the use of SR, which means access to SR is gained through SDM.

6.1.4 Deployment

Two concepts are important regarding the deployment of UbiCollab components:

UbiNode

A UbiNode is a mobile device which acts as a user's personal server. Typically, a UbiNode will run central UbiCollab services such as the SD-Manager and CI-Manager.

UbiNetwork

A UbiNetwork is a network of users sharing the same user management infrastructure. In most cases, UbiNetwork will have a special node called UbiHome, which runs shared services. This means that UbiHome acts as a sort of server within the UbiNetwork.

6.2 Tags in UbiCollab

This section looks at how the analysis and requirements found in previous chapters can be used to extend UbiCollab with support for social tagging. From the UbiCollab architecture and concepts introduced in the previous section, two main cases of tagging have been identified. These will be discussed in the following subsections.

6.2.1 Private tagging

Private or personal tags are tags added by a user that are not visible to other users. As can be seen from the introduction to the concepts of SD and SP, services are available in a user's SD once they have been installed. One approach to providing private tagging functionality would be to manage and store private tags within the user's SD. In other words, private tags could be added by the user to services present in his SD. A major probem with this approach is that installed services, represented by their SPs, are not necessarily permanently located in the user's SD. For example services that are never used could be removed. What happens to the services' tags in this case? To rephrase the problem; it is not favourable that all services that are to be tagged privately have to be installed in the user's SD.

A SR manages a repository of services within one UbiNetwork. Once a service has been registered, it is permanently in the SR, unless of course the service is not in use anymore. This means that service representations in the SR are not exposed to frequent installation and uninstallation such as in a user's SD. Thus a better approach would be to associate tags with the representation of services in the SR.

6.2.2 Collaborative tagging

While the previous section considered tags created and viewed privately, this category involves a community of users or everyone collaboratively tagging services. The main difference between collaborative and private tagging is that collaborative tags are shared between users. This means that when one user creates a tag to a service, the tag is automatically available to other users. UbiCollab supports communities by providing collaboration instances. This means that the basic mechanisms needed for community tagging are already in place. Public tagging is simply a sub-case of community tagging, where the collaboration instance is open to everyone. The connection between community tags and CIs can simply be implemented by using the CI ID.

For the same reasons as with private tagging, it is not appropriate to associate collaborative tags with a user's SD. For collaborative tags this would also mean that tags would be spread among different users' SDs, making tag sharing difficult. Thus, a more centralised solution is needed, which will be discussed further in Section 6.4.

Another difference between collaborative tagging and private tagging is that a connection to the tag creator is needed. This is needed to fulfil the requirement that more than one user can add the same tag to the same service. Also, the list of users who created a tag can be retrieved for tag-based navigation purposes. Again, this connection can simply be implemented by using a user ID.

6.3 Tag representation

From the previous sections, three relations between tags and other entities are identified

Service

Because a tag is applied to a service instance, there is a connection between tag and service. *ServiceURI* is the service identifier.

User

A tag is created and owned by a user, which means there is a connection between tag and user. Also, equal tags created by different users are treated separately. *UserID* is the user identifier.

Collaboration Instance

If a tag is created at community level, there is a connection between tag and collaboration instance. This relation can be seen as optional, since it doesn't apply for public and private level. *CiID* is the collaboration instance identifier.

In addition to the identified relations, a tag has the attributes *ID* and *name*. *ID* is a unique identifier for the tag, while *name* is the tag's text string representation.

The resulting ER diagram is shown in Figure 6.3, including cardinalities between the entities.

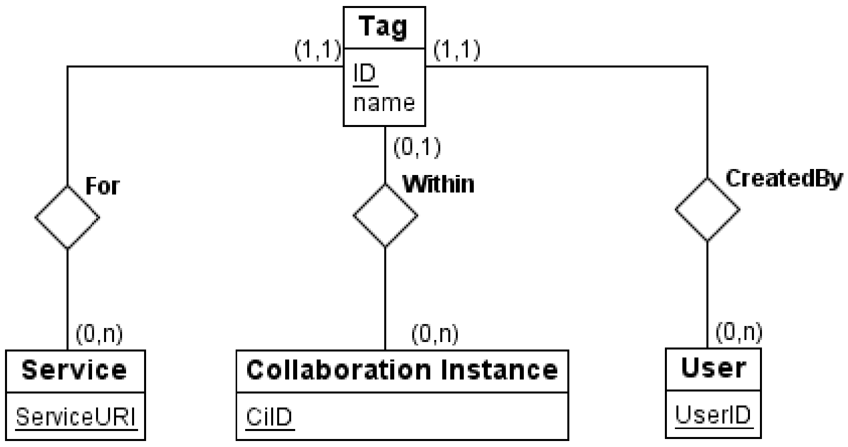


Figure 6.3: ER diagram of tag representation

6.4 Deployment

This section looks at how the tagging functionality can be deployed into one or more platform services.

6.4.1 Modularity

An important question that comes from the discussion so far is whether to extend existing UbiCollab services, such as SD-Manager and SR with tagging functionality. In order to keep UbiCollab modular and simple, which is an important part of a SOA, it is better to separate the tagging functionality into its own service or services. This solution provides a loose coupling between the tagging system and other UbiCollab components. Also, it helps avoid overloading existing UbiCollab services and also gives the possibility of having the tagging functionality as optional functionality.

6.4.2 Client and server

The choice of where to handle and store tags mainly relies on which part of the system the tags should be available to. Also, the problem of large amounts of tags needs to be considered, and how a centralised solution implies a potential bottleneck.

Private tags have the advantage of only needing to be available to the tag owner. This means that private tags can and should be handled locally, for example in the user's UbiNode. Public and community tags however need to be available to many users. Here it is possible to make use of the fact that services belong to and are registered in one UbiNetwork. This means that tags can be handled by UbiHome within a UbiNetwork rather than globally, which gives a distributed solution where each UbiNetwork takes care of its services' tags.

This suggests a separation of tagging functionality into two services, a server service running on UbiHome and client services that run on users' UbiNodes. The server service handles public and community tags, while the client service handles private tags.

In addition to the separation of tag visibility levels, the idea is that end user tools and applications communicate with the client service, rather than directly with the server service. When needed, the client communicates with the server service, for example when handling community or public tags.

One of the benefits of this solution is that it opens up for the possibility of local processing. For example the client and server can communicate in the background, without requiring the user to have a Graphical User Interface (GUI) running. As a scenario, consider a user coming into a room filled with several services. The client service could automatically get the services' public tags from the server service, anticipating that the user will want to view these.

6.4.3 Communication with other services

As has been talked about, there is a connection between the tag management part of the system and the SR. This is because each tag is connected to a registered service. Information about the service a tag is connected to can be retrieved by invoking appropriate functionality in SR through Service Discovery Manager. In a similar way, tag management is connected to user management and collaboration instance management, because each tag is connected to a user and possibly also a collaboration instance.

For all these connections between tag management and other parts of UbiCollab, there is the question of whether they should be handled by the tag management platform functionality. Alternatively, the connections can be "abstract" on the platform level, and then it's up to the end user tools and applications to integrate the parts. For the major part the latter should be the preferred choice, because the platform should not decide how the end user tool or application implements the integration. Another important argument is that couplings between modules of the platform should be minimalised.

6.5 Architecture

The overall architecture resulting from this chapter is shown in Figure 6.4. This shows the relations between the main components. The components are presented

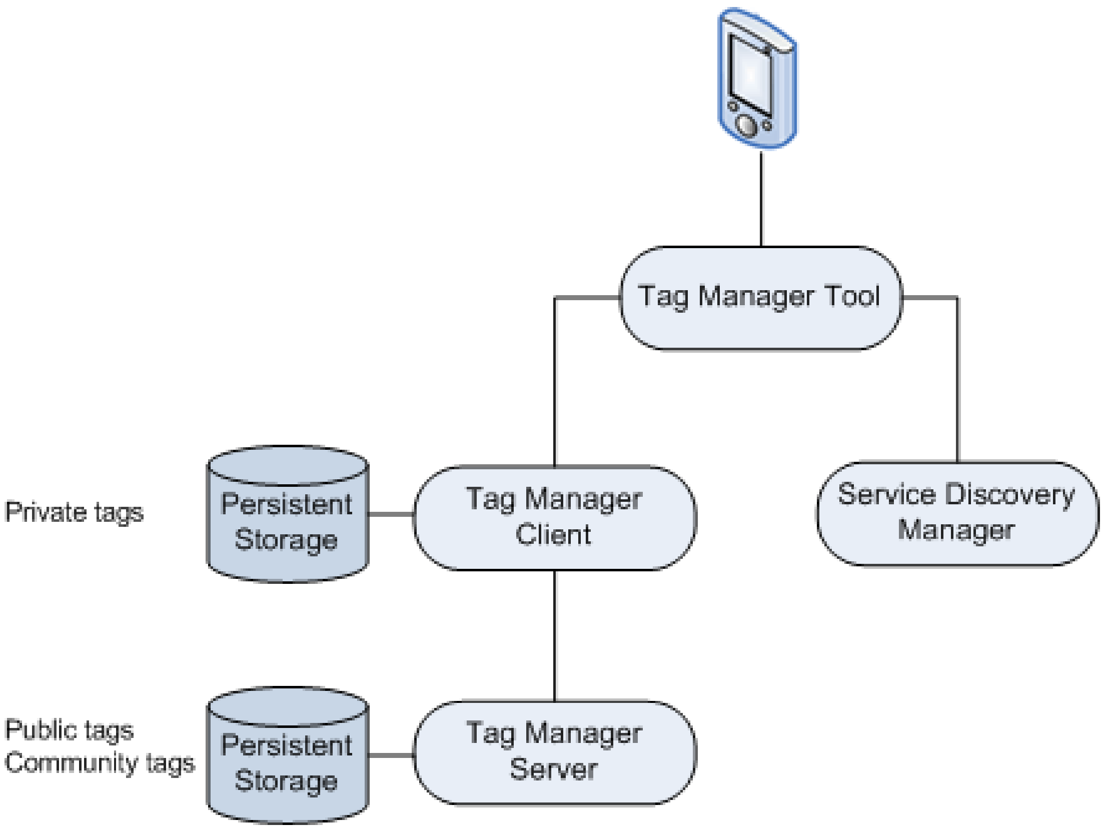


Figure 6.4: Tagging system architecture

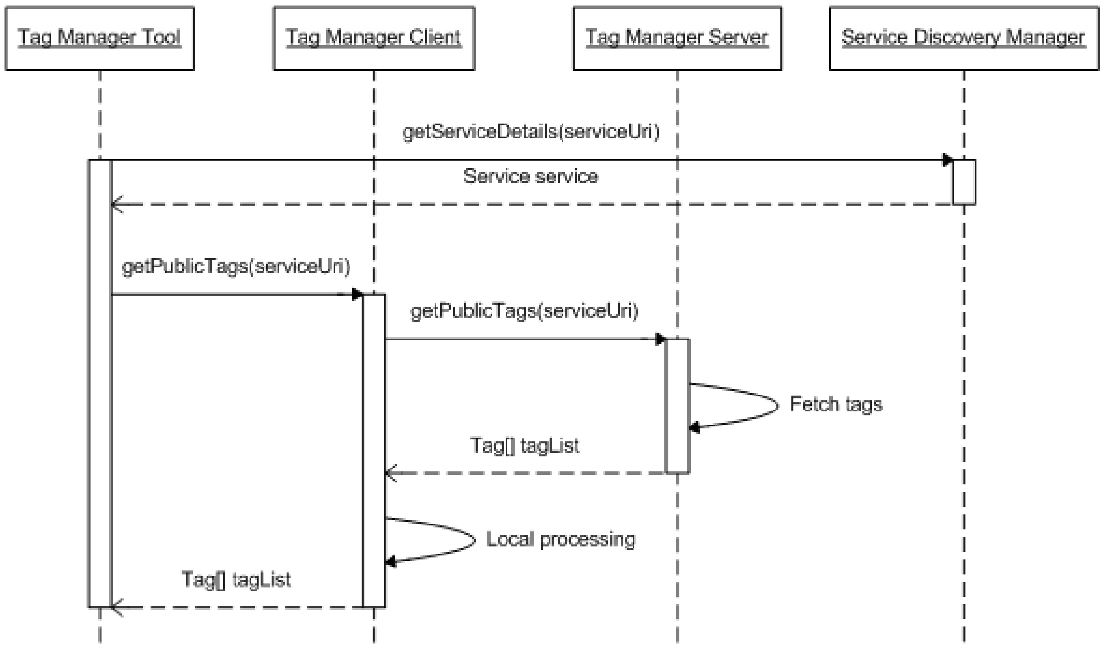


Figure 6.5: Component interaction

in the following subsections. For the platform services, APIs of the methods needed to implement the functionality are presented.

To illustrate how the components interact, a sequence diagram is shown in Figure 6.5. The diagram shows the interaction that could occur when **Tag Manager Tool** wishes to find a service's information and associated tags. Given a service URI, **Tag Manager Tool** queries **Service Discovery Manager** to get the service details. Next, **Tag Manager Tool** queries **Tag Manager Client** to retrieve the service's public tags. Because public tags are handled by the server, **Tag Manager Client** gets the public tags by calling **Tag Manager Server**, and does necessary local processing if any, before the list of tags is returned to the **Tag Manager Tool**.

6.5.1 Tag Manager Server

Tag Manager Server handles and stores public and community tags. The service is meant to act as a server for instances of **Tag Manager Client**. This means that the service should run on a shared node in UbiCollab, for example UbiHome.

6.5.1.1 API

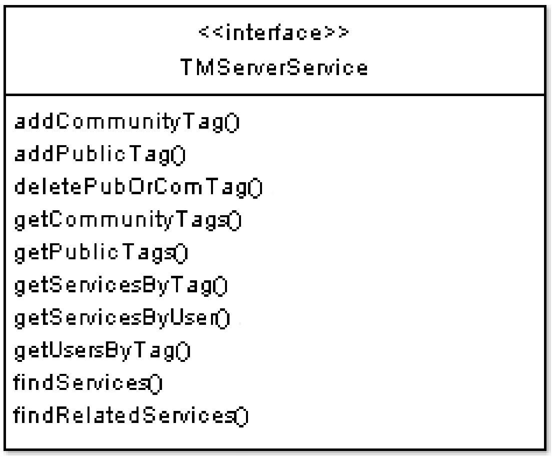


Figure 6.6: TM Server service API

The functionality provided by the service's API can roughly be divided into two categories, one for creating and removing tags and one for finding and retrieving tags, users and services. The API is shown in Figure 6.6.

The methods *addPublicTag* and *addCommunityTag* create new public and community tags respectively, while the method *deletePubOrComTag* provides the user with the possibility of deleting a public or community tag he has created.

Functionality for retrieving tags for a service is provided by the methods *getPublicTags* and *getCommunityTags*. The methods *getServicesByTag* and *getServicesByUser* provide two ways of finding services. The first method finds and returns all services that have been tagged with a certain tag, while the latter finds all services that have been tagged by a certain user. The method *getUsersByTag* finds and returns all users that have added a certain tag to the a certain service. Finally, two methods provide functionality for searching. *findServices* searches for services matching keywords, while *findRelatedServices* searches for services with tags similar to the tags of a certain service.

6.5.2 Tag Manager Client

Tag Manager Client is the local service which end user tools and applications should communicate with, rather than communicating directly with **Tag Manager Server**. The main reason for this is to open up for possible local processing in future implementations, as discussed in Subsection 6.4.2.

6.5.2.1 API

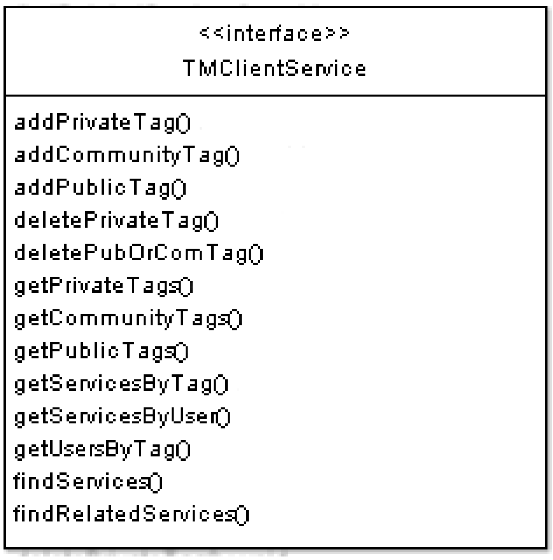


Figure 6.7: TM Client service API

In addition to the methods dervied from the server service, the client service's API provides functionality for handling and storing private tags. The API is shown in Figure 6.7.

The method *addPrivateTag* creates a new private tag. *deletePrivateTag* provides the user with the possibility of deleting a private tag he has created. Finally, the method *getPrivateTags* finds and returns private tags for a service.

6.5.3 Tag Manager Tool

The **Tag Manager Tool** component provides users or applications with access to the underlying platform components. To do this, the tool communicates with

Tag Manager Client and indirectly with **Tag Manager Server** through **Tag Manager Client**.

In addition to providing the tagging functionality, the tool combines and integrates other UbiCollab services. To retrieve service details, the tool communicates with **Service Discovery Manager**. An example of this was given in the sequence diagram earlier (see Figure 6.5). In a similar way, the tool needs to communicate with other UbiCollab services to retrieve user and CI information.

Finally, this component can be used as a demonstrator tool for demonstrating and evaluating the implemented solution.

Chapter 7

Implementation

This chapter describes the implementation of the social tagging system design solution presented in the design chapter. The platform services **Tag Manager Client** and **Tag Manager Server** were implemented and are presened with class diagrams, explanations of service methods, and discussions on implementation details. A **Tag Manager Tool** was also implemented, to provide access to the platform services' functionality and for demonstration purposes. A presentation of the tool's GUI is given, and various implementation details are discussed.

Section 7.1 presents the implementation of the platform services.

Section 7.2 presents the implementation of the tool.

7.1 Tag Manager platform services

This section presents the implementation of the server service and client service bundles.

7.1.1 OSGi

Both services were implemented as OSGi bundles for the Knopflerfish framework[1] which UbiCollab uses. This means that the services can be installed, started, stopped and uninstalled in the same way as other UbiCollab services, on devices running the framework. Knopflerfish is based on the Java programming language, which means that the bundles were implemented using Java. The bundles were named **TMClient** and **TMServer**, using the abbreviation TM to denote "Tag Manager".

Service functionality was provided as a web service interface by using the osgi-axis bundle in Knopflerfish. When this bundle is installed and started, it automatically takes care of publishing bundles' service interfaces as web service interfaces.

[1]https://www.knopflerfish.org/

7.1.2 Tag storage

The method used to store tags for both services is a MySql[2] database. For each service, a "Tag" table is used, with the attributes needed to represent a tag. For the client service, this is simply the tag name and the service URI. The server service uses additional fields for user and collaboration instance IDs. To represent public tags, the collaboration instance ID is set to the value "0".

7.1.3 Type representation

Several of the services' methods return tags, users or services. The natural way of implementing this is to represent a type with a class, and then return an instance or an array of instances of the class. However, the Knopflerfish osgi-axis bundle does not currently support the conversion (deserialising) of class instances to a suitable representation when returning data from a web service call. This means that service methods have to "manually" return an appropriate representation of types. To do this, strings of Extensible Markup Language (XML) were used. A consequence of this was that service methods returning types such as Tag or Tag[] had to be changed to String. An example of a list of tags is given below.

```
<TagList>
<Tag ServiceUri="http://nabaztag.no" UserId="56" CiId="45" Name="messaging"/>
<Tag ServiceUri="http://nabaztag.no" UserId="103" CiId="70" Name="friends"/>
</TagList>
```

7.1.4 TMServer bundle

This section describes the implementation of the Tag Manager Server service bundle, named **TMServer**. The service was implemented as an OSGi bundle, providing its functionality through a web service interface. The main responsibility of the service is to handle public and community tags by providing appropriate methods for creating tags and retrieving tags, users and services.

7.1.4.1 Class diagram

The bundle's class diagram is shown in Figure 7.1. The diagram shows the main classes and methods, and relations between classes.

The class **Activator** implements the Knopflerfish interface **BundleActivator**, thus implementing the methods *start* and *stop* which are called when the bundle is started or stopped. The main job of *start* is to register and publish the service interface represented by the interface **TMServerService**, while *stop* performs necessary resource cleanup.

[2]www.mysql.org

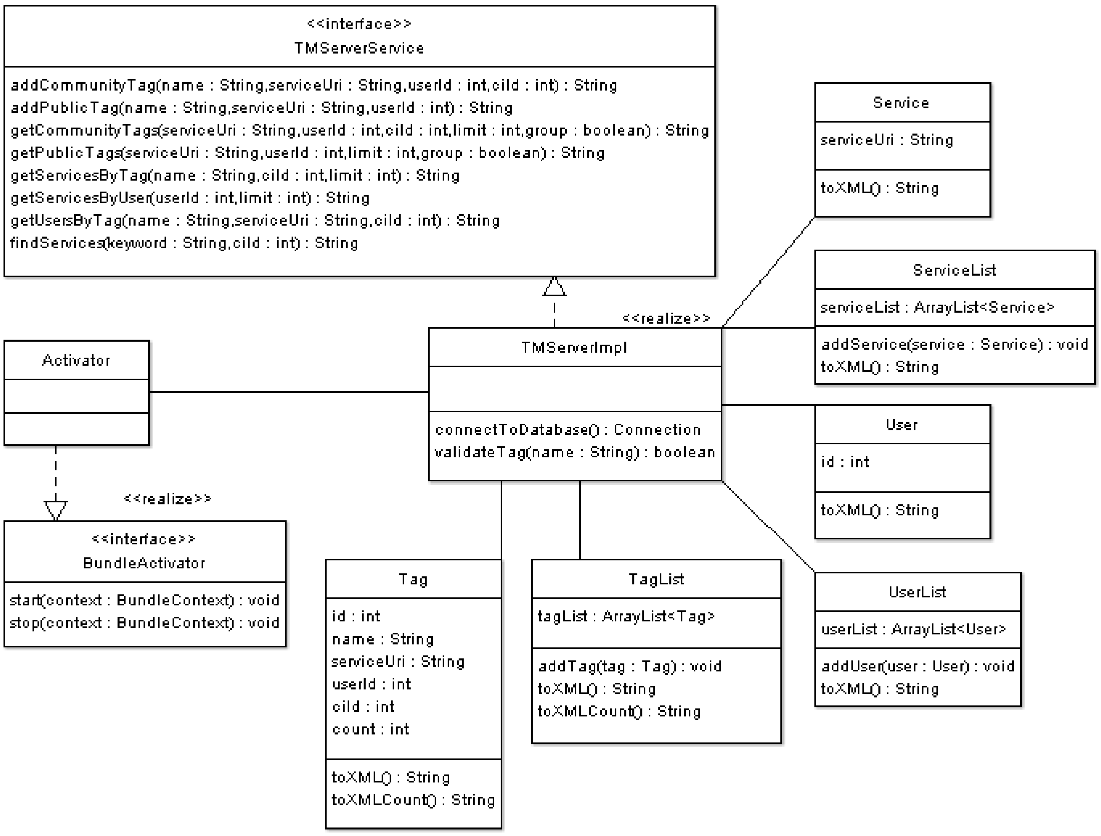

<<interface>>
TMServerService
addCommunityTag(name : String,serviceUri : String,userId : int,cild : int) : String
addPublicTag(name : String,serviceUri : String,userId : int) : String
getCommunityTags(serviceUri : String,userId : int,cild : int,limit : int,group : boolean) : String
getPublicTags(serviceUri : String,userId : int,limit : int,group : boolean) : String
getServicesByTag(name : String,cild : int,limit : int) : String
getServicesByUser(userId : int,limit : int) : String
getUsersByTag(name : String,serviceUri : String,cild : int) : String
findServices(keyword : String,cild : int) : String
Service
serviceUri : String
toXML() : String
ServiceList
serviceList : ArrayList<Service>
addService(service : Service) : void
toXML() : String
<<realize>>
Activator
TMServerImpl
connectToDatabase() : Connection
validateTag(name : String) : boolean
User
id : int
toXML() : String
<<realize>>
<<interface>>
BundleActivator
start(context : BundleContext) : void
stop(context : BundleContext) : void
Tag
id : int
name : String
serviceUri : String
userId : int
cild : int
count : int
toXML() : String
toXMLCount() : String
TagList
tagList : ArrayList<Tag>
addTag(tag : Tag) : void
toXML() : String
toXMLCount() : String
UserList
userList : ArrayList<User>
addUser(user : User) : void
toXML() : String

Figure 7.1: TMServer class diagram

Tags, users and services are represented internally with the classes **Tag**, **User** and **Service**. To convert an instance to XML, each class has a *toXML* method which returns a string of XML. For each class there is an associated class for containing a list of instances. When the *toXML* method is called on a list class, a string of XML representing the list is generated by calling the *toXML* on each of the list members. The classes **Tag** and **TagList** also have the method *toXMLCount* which is used in the case of duplicate tags being merged. In that case, the number of copies (count) of each tag is included.

TMServerService is the interface published and available through web service calls. It is is almost identical to the one presented in the design, except that the method for deleting tags was not implemented due to time constraints. Also, the method for finding related services was not implemented. The implementation of the interface is done by the class **TMServerImpl**. A presentation of the implementation of each method is given in the following subsection. In addition to these methods, **TMServerImpl** has methods for connecting to the database and for validating a tag.

7.1.4.2 Implementation of service methods

Following is a presentation of **TMServerService**'s methods. For all methods, errors due to missing or invalid parameters are handled by returning an appropriate message describing the error, for example “UserID missing”. Some of the methods have a “limit” parameter, which limits the number of objects (tags, services or users) to the given number.

addPublicTag

This method creates a new public tag with the specified name, associated with the given service URI and user ID. To create the tag, a database query is performed. The method returns a message with the result of the operation.

addCommunityTag

The implementation of this method is identical to *addPublicTags*, apart from the additional CI ID specifying which CI the tag belongs to.

getPublicTags

This method performs a database query to find public tags matching the service URI, and returns a list of tags. If the optional user ID parameter is specified, only public tags created by the given user are returned. The method also has a “group” parameter which decides if tags with the same name should be merged. If the parameter is true, duplicate tag names are returned as one, including the number of times the tag name occurred.

getCommunityTags

The implementation of this method is identical to *getPublicTags*, apart from the additional CI ID. Only tags within the specified CI are returned.

getServicesByTag

This method generates a list of services represented by service URIs, by performing a database query for services that have been tagged with the given tag name. The method also has a parameter for limiting services to a CI.

getServicesByUser

This method generates a list of services represented by service URIs, by performing a database query for services that have been tagged by the given user.

getUsersByTag

This method generates a list of users represented by user IDs, by performing a database query for users that have added the given tag to the given service. The method also has a parameter for limiting services to a CI.

findServices

Finally, this method implements a simple search. The method only accepts a single keyword, and finds services with tag names exactly matching the keyword. The services found can be limited to a CI by using the CI ID parameter. The method returns a list of services represented by service URIs.

7.1.5 TMClient bundle

This section looks at the implementation of the Tag Manager Client service bundle, named **TMClient**. The service was implemented as an OSGi bundle, providing its functionality through a web service interface. The service handles private tags, and acts as an intermediatery between tag management tools and the server service.

In the current implementation, the client service does not perform any local processing for public and community tags. This means that web service calls regarding public and community tags are simply passed on to the server service. Because the client and server services do not run in the same OSGI context, it is not possible for the client service to use local OSGI method calls to do this. Instead, a proxy had to be generated, which is a set of classes and methods for handling method calls to a remote web service. The proxy was generated using a tool in the axis-osgi bundle.

7.1.5.1 Class diagram

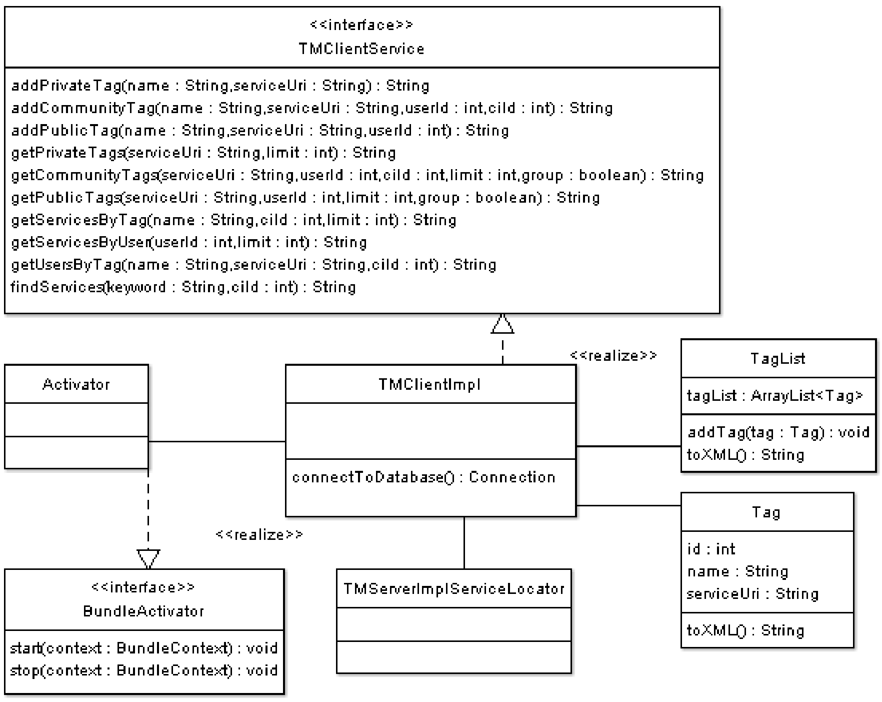

<<interface>>
TMClientService
addPrivateTag(name : String,serviceUri : String) : String
addCommunityTag(name : String,serviceUri : String,userId : int,cId : int) : String
addPublicTag(name : String,serviceUri : String,userId : int) : String
getPrivateTags(serviceUri : String,limit : int) : String
getCommunityTags(serviceUri : String,userId : int,cId : int,limit : int,group : boolean) : String
getPublicTags(serviceUri : String,userId : int,limit : int,group : boolean) : String
getServicesByTag(name : String,cId : int,limit : int) : String
getServicesByUser(userId : int,limit : int) : String
getUsersByTag(name : String,serviceUri : String,cId : int) : String
findServices(keyword : String,cId : int) : String
<<realize>>
Activator
TMClientImpl
connectToDatabase() : Connection
TagList
tagList : ArrayList<Tag>
addTag(tag : Tag) : void
toXML() : String
Tag
id : int
name : String
serviceUri : String
toXML() : String
<<realize>>
<<interface>>
BundleActivator
start(context : BundleContext) : void
stop(context : BundleContext) : void
TMServerImplServiceLocator

Figure 7.2: TMClient class diagram

The bundle's class diagram is shown in 7.2

Similar to the server service, a **Tag** class is used to represent a Tag, with an associated list class. The difference is that no method for including tag count is used, as duplicate private tags cannot exist. Classes for representing services and users are not needed, since all method calls that need to represent these types are passed on to the server service.

TMServerImplServiceLocator is the proxy class used to retrieve a server service object through which web service calls can be invoked.

TMClientService is the interface published and available through web service calls. Similar to the server service, the interface is identical to the design, except for delete methods not being implemented. The implementation of the **TMClientService** interface is done by the class **TMClientImpl**. A presentation of the implementation of each method is given in the following subsection. In addition to these methods, **TMClientImpl** has a method connecting to the database.

7.1.5.2 Implementation of service methods

Following is a presentation of **TMClientService**'s methods. Because all methods regarding public and community tags are passed on to the server service, these methods are not repeated here.

addPrivateTag

This method creates a new private tag with the specified name, associated with the given service URI. The tag is created through a database query.

getPrivateTags

This method performs a database query to find private tags matching the service URI, and returns a list representing the tags.

7.2 Tag Manager Tool

To provide access to the platform services' functionality, a Tag Manager Tool was implemented. This section describes implementation details of the tool, and presents the tool's GUI.

7.2.1 Implementation details

First, the following subsections look at some details of the implementation.

7.2.1.1 Microsoft .NET

The implementation of the tool was done using Microsoft's .NET platform and the programming language C#. The reason for choosing .NET is that making GUIs is easier compared to many other platforms and programming languages. Also, by using a different language than Java, the platform and programming language independency of the implementation is shown.

7.2.1.2 Combining platform services

To communicate with other UbiCollab platform services, web service references were created using a tool in Microsoft's Visual Studio programming environment. This automatically created the necessary functions for invoking methods on remote web services.

To provide the tagging system's functionality, a web service reference was created to the **TMClient** service. Through this reference, the application is able to invoke the **TMClient** service's methods.

To retrieve service details, the idea was to communicate with Service Domain Manager. However, the needed functionality to find and get a service's details based on the service URI had not yet been implemented. This meant that a reference directly to the Service Registry service had to be created. Through the reference, appropriate methods can be invoked to retrieve service details by specifying a service URI.

Finally, the tool integrates with the service discovery part of UbiCollab. By creating a web reference to the Service Discovery Manager service, the application can register as a client interested in discovered services. Whenever a service is discovered, the tool is notified.

The tool should also communicate with platform services regarding user and CI details. But as these services are not yet implemented, this was not possible. Instead, dummy values for user and CI names and IDs were used.

7.2.2 RFID reader

To discover services using an RFID reader, the tool **SDPluginRFID** needs to be run in parallell to **Tag Manager Tool**. The plugin has been developed as part of the service discovery implementation by a fellow UbiCollab team member [7]. It uses bluetooth to connect to the RFID pen. Whenever the pen reads an RFID tag, the tag data is registered by the plugin and passed on to **Service Discovery Manager**. At this point, **Tag Manager Tool** is notified of the discovery.

7.2.3 GUI

The tool's GUI is separated into three main tabs (sections). For each tab, an explanation of what the tab does is given accompanied by a screenshot with labels explaining what each part of the tab does.

7.2.3.1 Service

The **Service** tab displays a service's name, description and other information. In addition to service details, a list of the service's tags is shown. The level of

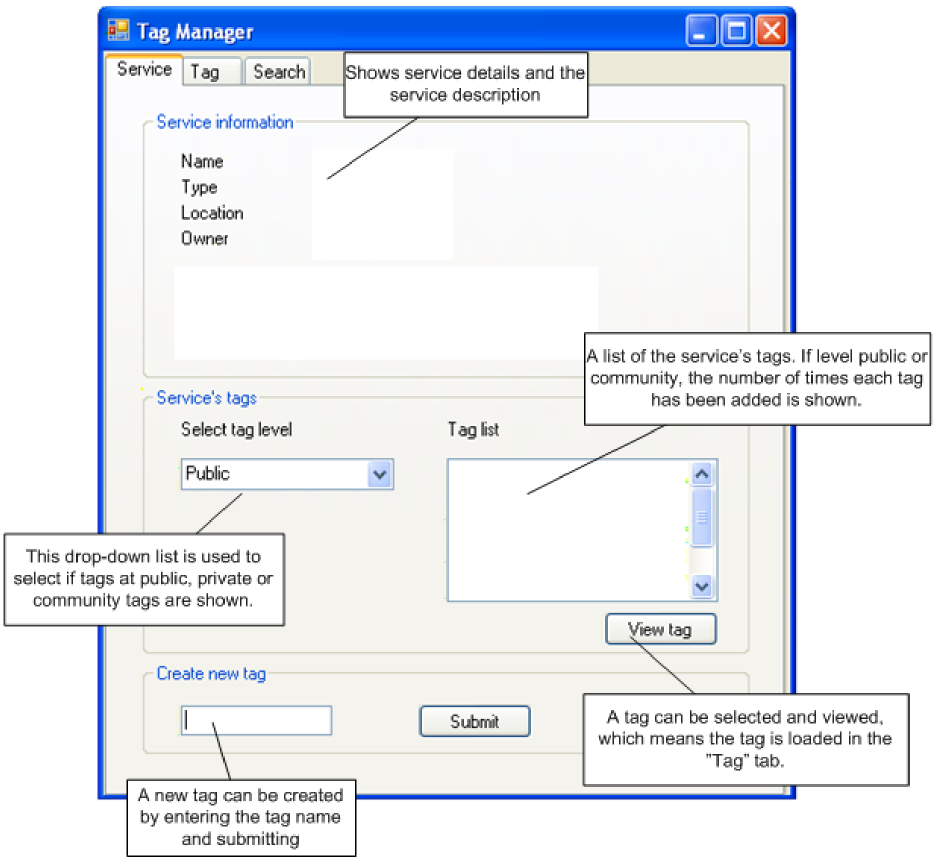

Tag Manager
Service
Tag
Search
Shows service details and the service description
Service information
Name
Type
Location
Owner
A list of the service's tags. If level public or community, the number of times each tag has been added is shown.
Service's tags
Select tag level
Tag list
Public
This drop-down list is used to select if tags at public, private or community tags are shown.
View tag
Create new tag
Submit
A tag can be selected and viewed, which means the tag is loaded in the "Tag" tab.
A new tag can be created by entering the tag name and submitting

Figure 7.3: Service tab

tags shown can be selected in a drop-down list. For each of the tags it is possible to select the action “view tag”, which will load the tag in the **Tag** tab. Finally, a new tag can be created by entering a tag name and submitting. The **Service** tab is shown in Figure 7.3.

7.2.3.2 Tag

When a tag is selected, the **Tag** tab displays the tag’s name and the name of the service which the tag belongs to. The main purpose of this part of the application is to show possible ways of providing tag based navigation. Therefore, a list of users who have added the tag to the service is displayed. When a user is selected, two more lists are shown. One displays other tags the selected user has added to the service. The other shows other services the selected user has tagged. In both cases it is possible to continue navigating by selecting items in the lists. The **Tag** tab is shown in Figure 7.4.

7.2.3.3 Search

The last part of the application is the **Search** tab. It provides a simple search feature, where a keyword can be entered and submitted, and a list of services

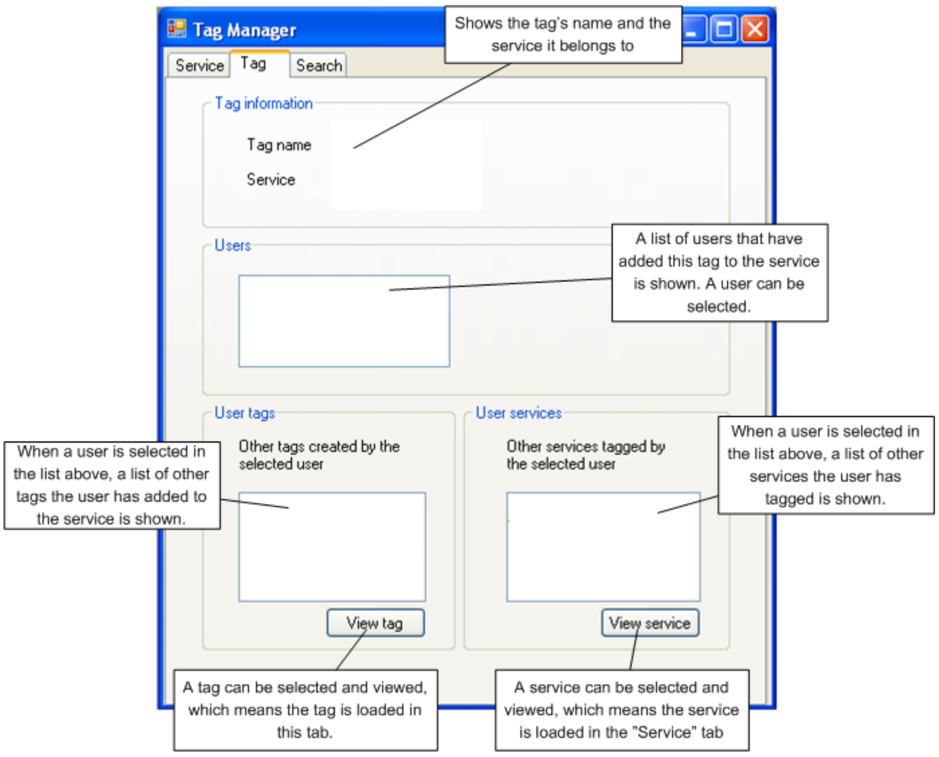
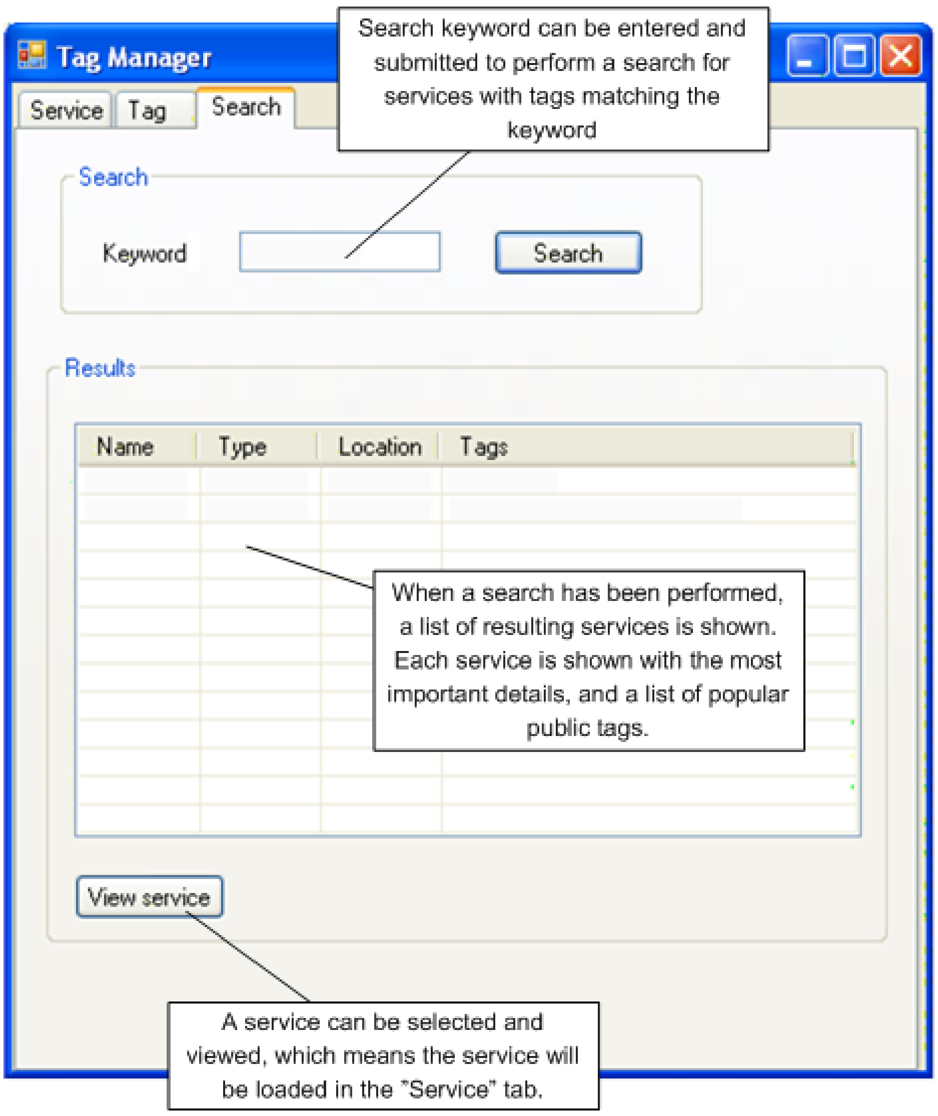

Tag Manager
Service
Tag
Search
Tag information
Tag name
Service
Users
User tags
Other tags created by the selected user
View tag
User services
Other services tagged by the selected user
View service
Shows the tag's name and the service it belongs to
A list of users that have added this tag to the service is shown. A user can be selected.
When a user is selected in the list above, a list of other tags the user has added to the service is shown.
When a user is selected in the list above, a list of other services the user has tagged is shown.
A tag can be selected and viewed, which means the tag is loaded in this tab.
A service can be selected and viewed, which means the service is loaded in the "Service" tab

Figure 7.4: Tag tab

with tags matching the keyword is shown. For each of the resulting services it is possible to select the action “view service”, which will load the service in the **Service** tab. The **Search** tab is shown in Figure 7.5.



Search keyword can be entered and submitted to perform a search for services with tags matching the keyword

When a search has been performed, a list of resulting services is shown. Each service is shown with the most important details, and a list of popular public tags.

A service can be selected and viewed, which means the service will be loaded in the "Service" tab.

Figure 7.5: Search tab

Chapter 8

Demonstration

This chapter first presents the demonstration of the implementation. The demonstration was made using scenarios, with the purpose of showing the functionality and possible uses of the tagging system. Also, the system was demonstrated informally to several members of the UbiCollab team.

Next, the design and implementation is evaluated. Strengths and weaknesses of the solution are discussed, and the platform services are compared to the requirements specification.

Section 8.1 demonstrates the implementation.

Section 8.2 evaluates the solution.

8.1 Scenarios

The functionality of the implementation is illustrated with several scenarios, each showing different features. Each scenario is accompanied by screenshots, to show how information is presented to the user and how interaction with the application occurs. Also, a walkthrough of component activation and communication between components is given. The bundle names **TMClient** and **TMServer** are used to denote the platform services, while Tag Manager Tool is abbreviated **TMTool**.

8.1.1 Service discovery

Katia has just walked into one of the reading rooms at her university. She's looking for a service to help her communicate with her family. She's carrying her PDA, which is running the TM Tool.

- TMTool has registered as a client with Service Discovery Manager

In the corner of the room is a device which looks like a rabbit. Curious to find out if the device could be of any use to her, she walks over to it. She uses her

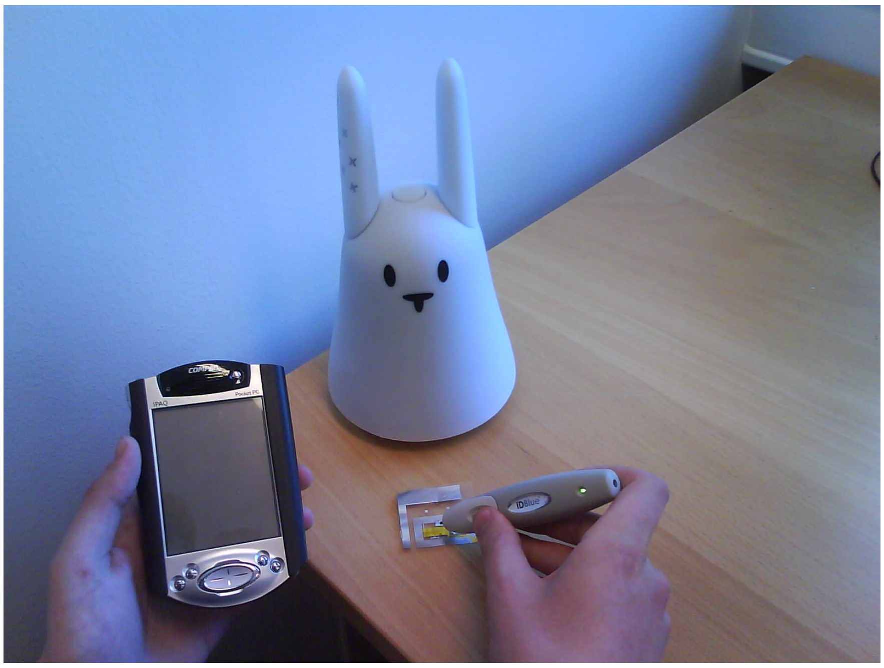


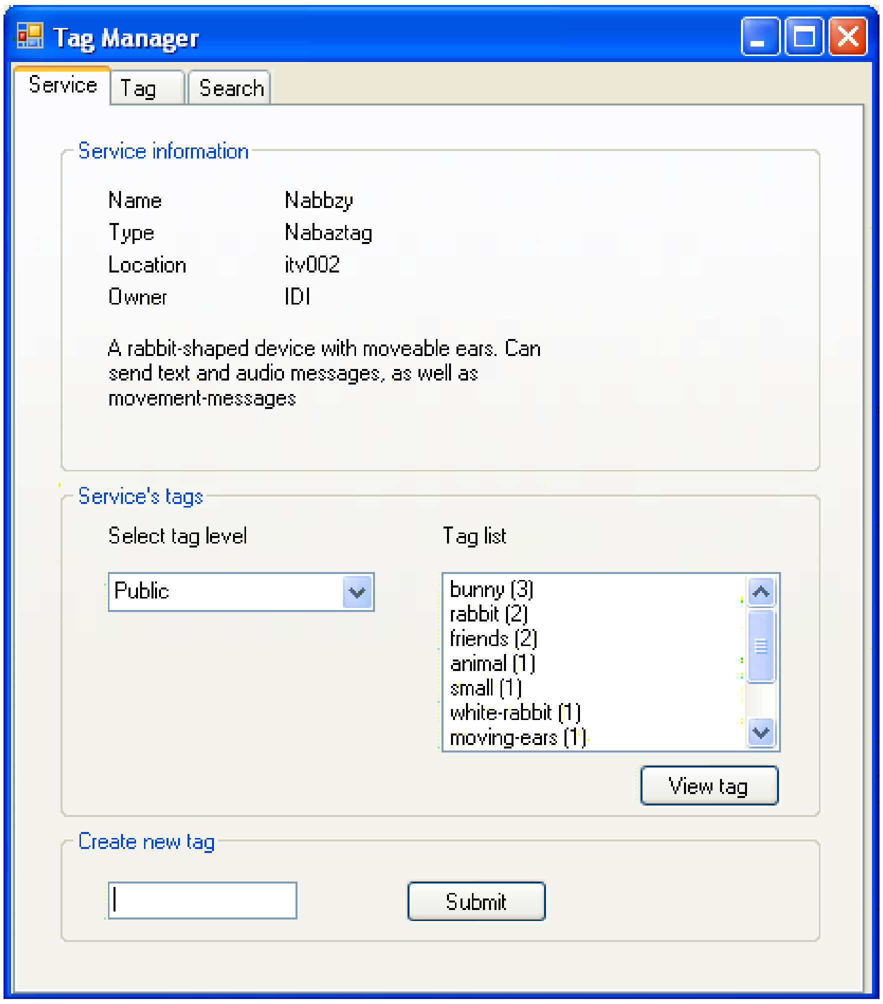
Figure 8.1: Service discovery with RFID

RFID pen to scan the label next to the device (see Figure 8.1)*, and after a few seconds the tag application notifies her that a service has been identified.*

- Service Discovery Manager notifies TMTool that a service has been discovered and gives the serviceUri "http://nabaztag.no/"
- TMTool displays a question asking whether to load the service or not

She confirms that she wants to view the service's details, and is presented with service name, description and other information. Also, a list of tags created by others is shown.

- TMTool calls the method *getServiceDetails* on Service Registry, with serviceUri "http://nabaztag.no/"
- Service Registry returns the service's details
- TMTool calls *getPublicTags* on TMClient with serviceUri "http://nabaztag.no/", userId 0, limit 10 and group true, on TMClient
- TMClient passes the method call on to TMServer
- TMServer returns a list of grouped public tags of the service, including tag count
- TMTool displays the service details and tag list



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

Select tag level: Public

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

Figure 8.2: Viewing the discovered service's details and tags

8.1.2 Creating tags

Katia is very happy with the Nabaztag rabbit, especially the moving ears. They are perfect for sending discreet messages. Wanting to share her experience with her collaboration instance of friends and family, she decides to create the tag "discreet-message".

- TMTool calls *addCommunityTag* on TMClient, with tag name "discreet-message", serviceUri "http://nabaztag.no/" and the userId of Katia and the collaboration instance ID of her community
- TMClient passes the call on to TMServer
- TMServer validates and stores the tag and returns a message saying the tag was succesfully added

She's so happy with the rabbit that she considers buying one to put in her apartment. She will look more into how she can get one when she comes home. Therefore she creates the private tag "to-buy", so she doesn't forget (see Figure 8.3).

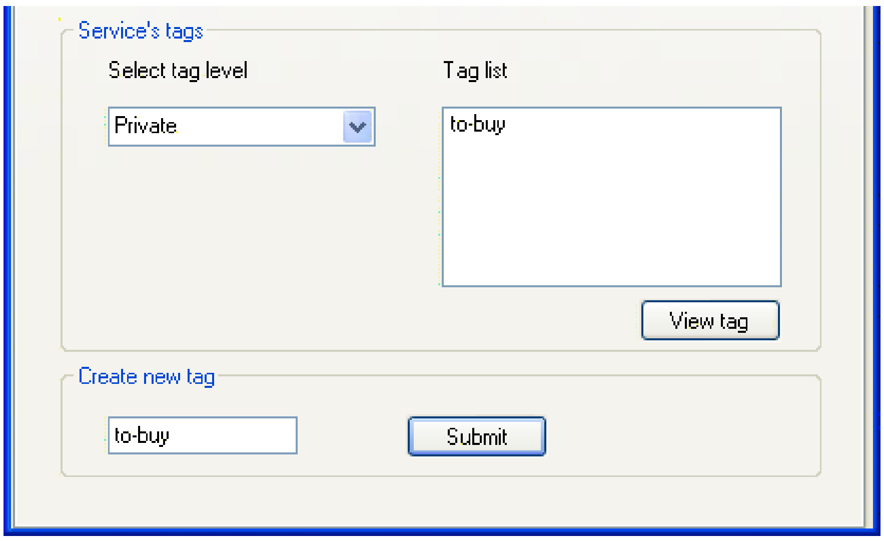


Figure 8.3: Creating a private tag

- TMTool calls *addPrivateTag* on TMClient, with tag name “to-buy” and serviceUri “http://nabaztag.no/”
- TMClient stores the tag and returns a message saying the tag was successfully added

8.1.3 Tag-based navigation

John is looking through the list of tags for a service he found, but the list is rather large and cumbersome to read. Therefore he chooses to view tags only within his collaboration instance of fellow students, whom he also thinks are more likely to share his opinions.

- TMTool calls *getCommunityTags* on TMClient. Parameters are serviceUri “http://www.ntnu.no/bulb”, userId 0, limit 10, group true, and the collaboration instance ID
- TMClient passes the call on to TMServer
- TMServer returns a list of grouped community tags for the service, including tag count
- TMTool displays the tags

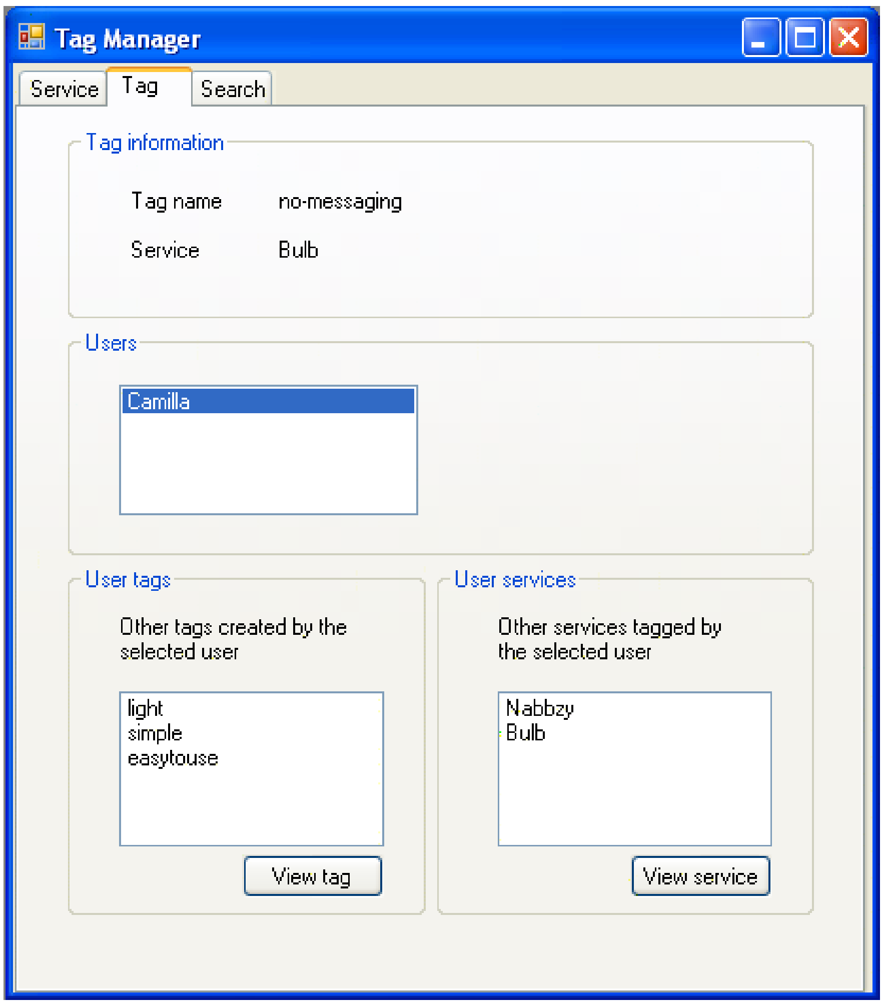
One of the tags is “no-messaging”. Wanting to find a service with messaging capabilities, he selects the tag, and a list of users who have added this tag is shown.

- TMTool calls *getUsersByTag* on TMClient, with tag name “no-messaging”, serviceUri “http://www.ntnu.no/bulb”, limit 10 and the collaboration instance ID

- TMClient passes the call on to TMServer
- TMServer returns a list of users that have added the tag “no-messaging” to the service
- TMTool displays the users

One of the users who added the tag is his friend Camilla. Selecting her name, a list of other services tagged by her is shown (see Figure 8.4). *It turns out that one of the services is exactly what he’s looking for.*

- TMTool calls *getServicesByUser* on TMClient, with the userId of Camilla and limit 10
- TMClient passes the call on to TMServer
- TMServer returns a list of services that have been tagged by Camilla
- TMTool displays the services



Tag name no-messaging

Service Bulb

Camilla

Other tags created by the selected user

light
simple
easytouse

Other services tagged by the selected user

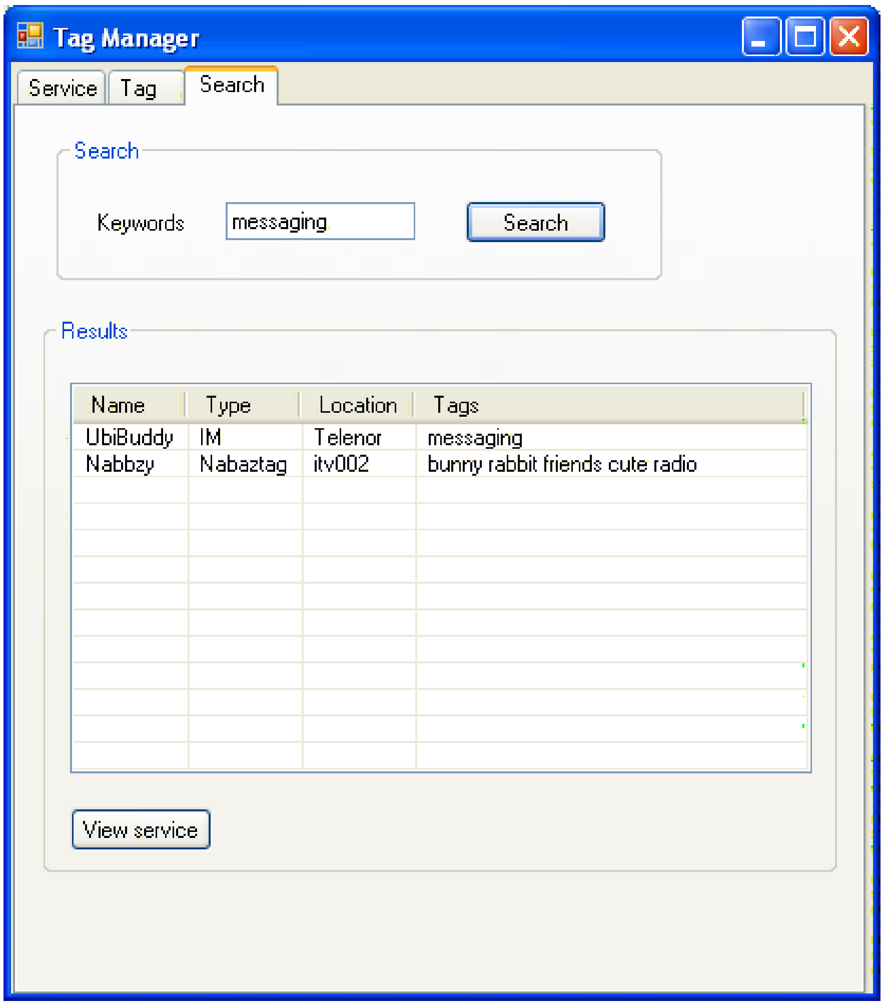
Nabbzy
Bulb

Figure 8.4: Tag-based navigation

8.1.4 Searching

Paul is looking for a nearby messaging service for communicating with a colleague. He doesn't know about any or where to look, so he uses the search feature of Tag Manager. He uses the keyword "messaging", which gives him two services (see Figure 8.5). *One of them looks promising, so he installs it in his service domain and publishes it to the collaboration instance where both he and his colleague are members.*

- TMTool calls *findServices* on TMClient, with keyword "messaging"
- TMClient passes the call on to TMServer
- TMServer finds and returns a list of services with the tag "messaging"
- TMTool displays the list of services



Keywords: messaging

Name	Type	Location	Tags
UbiBuddy	IM	Telenor	messaging
Nabbzy	Nabaztag	itv002	bunny rabbit friends cute radio

Figure 8.5: Searching

8.2 Evaluation of design and implementation

In this section the design and implementation of the platform services and the tool is evaluated. Strenghts and weaknesses of the solution are highlighted. Finally, possible improvements are suggested.

8.2.1 Platform services

In Table 8.1 and Table 8.2 the requirements from the requirements specification chapter are listed, together with the current status of the requirements in the implementation. Most of the requirements were implemented and fulfilled by the platform services. But due to time constraints, some were not implemented or only partially implemented.

Suggestions based on underlying ontologies (FA2) were not implemented. This is mostly because there currently is no ontology management in UbiCollab, making it hard to fulfil the requirement. The requirement for deleting tags (FA3) was not seen as critical and therefore not prioritised.

All requirements for browse and search functionality were implemented, except the possibility of finding related services (FB8). Also, only a limited search was implemented (FB9), supporting a single keyword. This was mainly due to limited resources, and the focus of this project not being on advanced search techniques.

The resulting platform services provide the functionality through a web service interface, supported by OSGi's web service publishing mechanisms. A very important consequence of this is that the interface provided is platform independent. This is shown by using a different programming language/platform in the tool implementation.

The platform services also fulfil the key aspect in SOA of being modular and loosely coupled with other components, meaning that there are no dependencies between the services and other UbiCollab platform services. First this means that the tagging functionality is optional. Second, the tagging services can be combined with other components of UbiCollab in any kind of way. A final and important consequence is that the platform services are not affected by changes made to other components.

8.2.2 Tool

A Tag Manager Tool was implemented, to give an example of a tool using the platform services and show the tagging functionality. The tool showed all functionality of the platform services, and demonstrated many of the ideas presented throughout this report. Demonstration of the tool to other UbiCollab team members resulted in positive feedback.

Although the tool served well for the purpose of demonstration, a limitation is that it is not easily integrated with other tools and applications. For example it could be natural to integrate it with a service domain tool, so that tagging support can be given when searching for and installing services in the service domain. In this case, the service domain tool will have to reimplement what is done by the tag manager tool. A better solution would be to implement a tag manager tool as a set of programming libraries/APIs, which could then be used when integrating tagging support in other tools and applications.

When implementing the tool, the challenge of providing a user friendly GUI became apparent. Positive feedback was given by other UbiCollab team members, but it was clear that the added complexity of separate levels of tagging could be hard to understand and use.

8.2.3 Possible improvements

Although a database was used in the implementation of the platform services, other storage methods are possible and could be considered in future implementations. For the server the additional performance and flexibility of a database is needed due to large amounts of tags. However on the client, it could be advantageous to use a simpler approach putting less requirements on the client. For example one could simply use the file system, with tags stored as files using XML.

The requirement for finding related services (FB8) presents a challenge in finding a suitable technique for comparing services and which criteria should be used. The number of tags matching, the tag count of the tags matching and the number of users that have tagged both are just some criteria that could be used when comparing two services. For the search requirement (FB9), the challenge is in finding a suitable and effective search technique, especially for searches using several keywords. Both cases could be looked at in future improvements, to improve the search mechanisms of the system.

In future implementations, Tag Manager Tool should communicate with Service Discovery Manager, rather than directly with Service Registry. This was part of the design solution, but due to missing functionality in Service Discovery Manager at the time of implementation meant that direct communication with Service Registry had to be implemented.

ID	Description	Status
FA1	It should be possible to create new tags for service instances	Implemented
FA2	Tag name suggestions should be given based on the underlying service ontologies	Not implemented
FA3	A tag should be possible to remove by the user who created it	Not implemented
		Continues next page

ID	**Description**	**Status**
FA4	A tag's visibility level should be possible to set when creating it	
FA4.1	It should be possible to set the tag as private	Implemented
FA4.2	It should be possible to set the tag as public	Implemented
FA4.3	It should be possible to set the tag as visible to one or more communities	Implemented by requiring the tag to be added once to each community
FA5	Different users should be allowed to create the same tag to the same service. The tags should be treated as unique	Implemented
FA6	A user should be prevented from creating the same tag more than once for a service, within the same visibility level. For communities, this means once per community	Implemented
FA7	Public and community tags should not be allowed to contain spaces and should not exceed a maximum length	Implemented
FA8	It should be possible to query the system for a service's tags	
FA8.1	It should be possible to retrieve tags at level public, private or community	Implemented
FA8.2	It should be possible to retrieve only a part of the resulting tag set	Not implemented
FA8.3	It should be possible to retrieve only the most popular tags in the resulting tag set	Implemented with the "limit" parameter, limiting tags to the given number of the most popular tags
FA8.4	It should be possible to find the number of times each tag name has been added at public or community level	Implemented with the "group" paramater. This merges and counts duplicate tags
FA9	The platform must provide a suitable, platform independent, interface to the platform functionality	Implemented as a web service interface

Table 8.1: Evaluation of basic tagging requirements

ID	**Description**	**Status**
FB1	It should be possible to query the system for the users who have tagged a service with a given tag	Implemented
FB2	It should be possible to query the system for other services tagged by a given user	Implemented
FB3	It should be possible to find who has added a tag	Implemented
FB4	It should be possible to find which other tags a user has added	Implemented
FB5	It should be possible to find which other services a user has tagged	Implemented
FB6	It should be possible to retrieve services that have been tagged with a given tag	Implemented
FB7	It should be possible to search for services by keywords.	A simple search is implemented, where a search can be performed using only a single keyword
FB8	It should be possible to find services that are similar to a given service, based on the services' tag sets.	Not implemented

Table 8.2: Evaluation of search and browse requirements

Chapter 9

Conclusions

This chapter concludes the work done in this project. First the contributions of the project and the fulfilment of the project goals is discussed in Section 9.1. Next, an overall evaluation of the work and the work process is given in Section 9.2. Finally, Section 9.3 presents suggestions for future work.

9.1 Contributions

The work done in this project has been a contribution to ASTRA and UbiCollab, and should also be of value to similar projects. The main goal of the project was to design and implement a framework for social tagging of services in a ubiquitous computing context. The goal was further divided into subgoals in the problem elaboration:

- **Subgoal 1**: Identify design space of social tagging and define design choices
- **Subgoal 2**: Identify mechanisms for tag based navigation and search
- **Subgoal 3**: Define requirements for a social tagging system
- **Subgoal 4**: Propose a design solution in the form of services and tools, extending the UbiCollab platform
- **Subgoal 5**: Implement, demonstrate and evaluate a prototype of the solution

Important design issues for tagging systems were identified and analysed in the problem analysis chapter, with a focus on how the choices made are affected by the problem context. Especially the physical nature of services embedded in devices was identified as an important difference compared to other tagging systems. This lead to the conclusion that physical access to tags should be provided. Also, a strong emphasis was put on providing community tagging. Although supporting community tagging is not entirely new, most tagging systems put little emphasis on group or community aspects. Often, such features are mainly for finding other users and connecting. This work has focused on providing three different visibility levels for tags; private, community and public. The community level is especially important, as this should help improve the quality and relevance of tags within a community. By viewing tags within one of his communities, the user is more likely to find what he is looking for. The analysis of design issues lead to a set of design choices, thus fulfilling *subgoal 1*.

For *subgoal 2*, the problem analysis discussed how the basic tagging mechanisms can be used. Relations between tags, services and users were identified, and how these relations can support tag based navigation. Also, searching mechanisms were discussed, and interaction with tags through physical access to services.

The problem analysis resulted in a requirements specification for a social tagging system, fulfilling *subgoal 3*. The requirements covered both basic tagging functionality and the use of the functionality for browsing and searching. The focus of the requirements was on the platform level, in other words how the UbiCollab platform can provide common functionality needed for end user tools and applications. The requirements were presented with Use Case diagrams, textual Use Cases and a detailed list of requirements.

For *subgoal 4*, a design solution was proposed to fulfil the specified requirements, in the context of UbiCollab. This meant that important UbiCollab concepts had to be discussed, and how the solution could be integrated with the concepts and existing services of UbiCollab. The design separates tagging functionality into two platform services, a client and a server service. This allows for private tags to be stored locally, while community and public tags are shared through the shared server service. Several advantages were shown with this solution, which enable a wider specter of possibilities for future implementations. In addition, the design consists of a tool for providing access to the platform services.

A prototype of the proposed design solution was implemented. This consisted of platform services implemented as OSGi service bundles, conforming to UbiCollab's SOA. The services can be integrated with other parts of the UbiCollab platform, and the APIs are provided through web service interfaces which can be used by end user tools and applications. A demonstrator tool was also implemented, which demonstrates the platform services' functionality. The demonstration was given with scenarios, showing possible uses of the system. This included the basic features of creating tags, as well as using tags for navigating and searching services. Finally, both strengths and weaknesses of the solution were pointed out in the evaluation, and suggestions for improving the solution. This means that *subgoal* 5 was also fulfilled.

9.2 Evaluation

The previous section showed that this work has accomplished the goal of creating a framework for social tagging of services in a ubiquitous computing context. The value of the work has been evaluated through a demonstration using scenarios which illustrate cases where the tagging system helps users to find and understand services. The solution was also sucessfully integrated with UbiCollab, extending the UbiCollab platform with social tagging support and conforming to the SOA's key aspects of modularity and simplicity.

The main focus of the work was narrowed down to how a social tagging system can be used to create a folksonomy for services. Thus, the proposed solution has the limitation of only considering this type and purpose of tags. The problem elaboration identified several other ideas and purposes of tags. Examples are evaluative tags (ratings), general comments, and tags pointing to resources. In hindsight, a solution which is more general and thus combines the different types of tags should maybe have been considered. Though, such a solution could easily become so complex that the advantage of simplicity in social tagging is lost. In any case, adding support for other tag types would not mean that the tagging system would have to be completely redesigned, as a major part of the analysis and decisions done in this work should apply in the case of other tag types.

A regret with the chosen work process was to start the implemention at a relatively late stage in the project. When starting to implement, many new ideas were found which helped get a better understanding of the problem and improve the analysis and requirements specification. This suggests that an iterative approach could have been better, rather than the classical waterfall model.

An important part of the work was to integrate the design and implementation of the tagging system with other parts of UbiCollab. This involved collaborating with other UbiCollab team members. Although the collaboration throughout the project could have been better, a clear improvement was seen towards the end. This resulted in successful integration of the tagging system with other components such as Service Discovery Manager and Service Registry. Especially the integration with Service Discovery Manager was successful and valuable, as this enabled the implementation and demonstration of physical access to tags.

9.3 Further work

In future work alternative uses of social tagging should be considered and how they can be incorporated in a tagging system. One of the main challenges is most likely to avoid the tagging mechanisms becoming too complex. Combining several types of tags could easily confuse users and discourage them from using the tagging system, which would obviously reduce the system's value.

The proposed design of architecture uses a client service as an intermediary between end user applications and the server service. This enables several future improvements, based on the possibility of local processing in the client service. Possibilities such as background processing and Peer-to-peer (P2P) communcication should be considered to further improve the tagging system.

One of the problems with the current implementation of the tagging services is that privacy and security issues are not handled. For example, an application can submit a tag giving any user and collaboration instance identifier. The tagging system does not validate the correctness of either identifiers. In future implementations applications should be authenticated, ensuring that the user identifier is valid and that the user is only given access to create and read tags within collaboration instances he is a member of.

This work focused on tagging of service instances, arguing that tagging of both service instances and types would make the system too complex for the user. But as suggested, this does not mean that generic tags are not of any value. An interesting task could be to look at tags in relation to ontology management and how instance tags can be used to deduce service type tags.

Chapter 10

References

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

[2] M. Bell, M. Hall, M. Chalmers, P. Gray, and B. Brown, 'Domino: Exploring mobile collaborative software adaptation', *Pervasive Computing* **Vol. 3968** (2006), pp. 153–168.

[3] B. A. Farshchian and M. Divitini, 'Ubicollab - white paper', *IDI Technical Report 07/07* (2007). `http://mediawiki.idi.ntnu.no/wiki/ubicollab/index.php/UbiCollab:Publications`.

[4] M. Fitzpatrick, *Tagging tokyo's streets with no name* (Published online, 2007). `http://technology.guardian.co.uk/weekly/story/0,,2075537,00.html`.

[5] S. Golder and B. A. Huberman, 'The structure of collaborative tagging systems', *The Structure of Collaborative Tagging Systems* **Vol. 32** (2006), pp. 198–208.

[6] M. Guy and E. Tonkin, *Folksonomies: Tidying up tags?*, in *D-Lib Magazine* (Corporation for National Research Initiatives, 2006).

[7] K.-S. Johansen, 'User-centered and collaborative service management in ubicollab: Design and implementation', *Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway* (Spring 2007).

[8] T. W. Malone, K.-Y. Lai, and C. Fry, 'Experiments with oval: A radically tailorable tool for cooperative work', *ACM Transactions on Information Systems (TOIS)* **Vol. 13** (1995), pp. 177–205.

[9] C. Marlow, M. Naaman, D. Boyd, and M. Davis, 'Ht06, tagging paper, taxonomy, flickr, academic article, to read', in *Proceedings of the seventeenth conference on Hypertext and hypermedia* (ACM Press, 2006), pp. 31–40.

[10] A. Mathes, *Folksonomies - cooperative classification and communication through shared metadata* (Published online, 2004). `http://www.adammathes.com/academic/computer-mediated-communication/folksonomies.html`.

[11] O. H. Nygård, 'Scenario and system specification', *D.2. ASTRA EU-IST29266 Technical Deliverable* (2007).

[12] B. Schneiderman, B. B. Bederson, and S. M. Drucker, 'Find that photo! interface strategies to annotate, browse and share', *Communications of the ACM* **Vol. 49** (2006), no. No. 4, pp. 69–71.

[13] S. Sen, S. K. Lam, A. M. Rashid, D. Cosley, D. Frankowski, J. Osterhouse, F. M. Harper, and J. Riedl, 'tagging, communities, vocabulary, evolution', in *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work* (ACM Press, 2006), pp. 181–190.

[14] G. Smith, *Atomiq: Folksonomy: social classification* (Published online, 2004). `http://atomiq.org/archives/2004/08/folksonomy_social_classification.html`.

[15] J. Trant and B. Wyman, *Investigating social tagging and folksonomy in art museums with steve.museum*, in *Proceedings of the WWW 2006 Collaborative Web Tagging Workshop* (2006).

[16] T. V. Wal, *Explaining and showing broad and narrow folksonomies* (Published online). `http://www.personalinfocloud.com/2005/02/explaining_and_.html`.

[17] Wikipedia, *Flickr* (Published on Wikipedia, 2006). `http://en.wikipedia.org/wiki/Flickr`.

[18] H. Wu, M. Zubair, and K. Maly, 'Harvesting social knowledge from folksonomies', in *Proceedings of the seventeenth conference on Hypertext and hypermedia* (ACM Press, 2006), pp. 111–114.

[19] X. Wu, L. Zhang, and Y. Yu, 'Exploring social annotations for the semantic web', in *Proceedings of the 15th international conference on World Wide Web* (ACM Press, 2006), pp. 417–426.