Abstract

Shared displays are important tools for promoting collaboration. Ubiquitous computing presents new requirements for the design of shared display systems. Contextualisation of information at shared displays is becoming more important.

The ability to rapidly create shared display systems is motivated by the fact that shared displays play central roles in collaboration. Low level implementation issues, common to shared display systems can be an obstacle for this. A toolkit for creation of such systems is therefore needed to provide basic shared display functionality to developers.

This master thesis presents a toolkit for creating shared display applications on UbiCollab, a platform supporting collaborative work in ubiquitous environments. The work shows the development of the toolkit and how the toolkit can be used to create a shared display system. The toolkit takes advantage of the opportunities the UbiCollab platform provides on contextualisation of information.

Preface

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

The master thesis contains work done on creating a toolkit for development of shared display systems on the UbiCollab platform, DISCOlab. The main results of this thesis are the report, a prototype implementation of the DISCOlab toolkit, and demonstration system using the prototype.

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

Trondheim, June 14, 2005

Carsten Andreas Heitmann

Contents

List of Figures

Chapter 1

Introduction

Shared displays have recently gained a growing attention [OPCR03a]. To replace conventional shared displays (paper-based) used by a community, eg. bulletin boards, timetables, whiteboard, etc., with electronic shared displays can potentially overcome the limitation of paper based artifacts, enrich the number of services offered to the community, improve participation to community life and support new forms of cooperation.

1.1 Shared display

A shared display provides two or more people with the same view of some information, like a document. Often the shared display has tools for manipulating the information, for example by drawing on it. Shared display applications can roughly be divided into two categories. Single display groupware that lets people gather around a single physical, shared display, and lets them input sequentially or simultaneously to the device. The other category is software that distributes a shared display onto several physical devices, and allows people to be distributed and see the same display [Bak04].

To share information at shared displays you need an entity that shares the information and an entity that displays the information. The entities can be on the same physical device. This depends on whether the shared display should be distributed or co-located.

To enhance collaboration, shared displays are important tools. Ubiquitous computing opens for even more possibilities. Despite a growing number of systems, some of which have also been tried out in real settings, the wide-spread adoption of these systems is far to come. One of the problems is high requirement for functional and technical flexibility, ie. these systems must be able to grow with their community, both in terms of new services that might be needed and in terms of new technical possibilities, for example, for promoting interaction by using new mobile devices [DF04]. To address this point we want to study the integration of shared displays into a platform

that offers support for collaborative work in a ubiquitous environment. To support easy development of this systems, we want to focus on the development of a toolkit.

In [Gre04], Greenberg points out that in the domain of groupware there is a lack of tools that support development of applications, unlike for example the user interface domain where there for decades have been toolkits available for building GUI applications in a rapid and easy way. Because of this, a groupware developer has to spend a considerable amount of work on low-level implementation issues such as network protocol implementations, worrying about distributed systems issues, etc. The result is that groupware applications become to hard to do, that developers spend their energy in low-level implementation issues, ending up with a groupware having low functionality and usability.

The UbiCollab platform is developed to support collaboration in a ubiquitous computing environment. The UbiCollab platform consists of several platform services that handle the management of collaborative efforts, people, resources, presence, privacy and location. Shared display systems are important tools in collaboration. To provide support for collaborating parties, contextualising the information is important, so that participating members are aware of each other and can obtain important information about the context the collaboration is done in. UbiCollab provides such information and creating shared display systems on top on UbiCollab is therefore highly interesting for improving the collaboration support.

An review of shared displays will be given in chapter 2.

1.2 Ubiquitous computing and groupware

Ubiquitous computing is a relatively new research area. The field is constantly undergoing changes as new techniques and technology for ubiquitous computing are developed. The technology is in an early stage and standards are constantly changing. This project is focusing on ubiquitous computing as a mean to enhance collaboration. Collaboration via groupware is a well studied field and many different applications are available for the users. However in a ubiquitous or mobile environment, few applications for collaboration are developed.

The term ubiquitous computing was first coined by the late Dr. Mark Weiser [Wei93]:

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

The term groupware was defined by Ellis as [Ell99]:

> Computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment.

The term groupware refers to software that can be used by a group of people, collaborating. The users of the groupware can be co-located or not co-located.

UbiCollab is a platform for supporting group collaboration in a ubiquitous environment. The vision and idea of UbiCollab is described in an article by Divitini et al. [DFS04].

1.3 Research focus and project goals

The goal of this project is to design and implement a toolkit for developing shared display systems that shall run on the UbiCollab platform. The toolkit is to be called DISCOlab, abbreviated from shared DIsplay System for COllaborative work. Shared display systems that run on the UbiCollab platform shall have access to the information that the services at the platform provide. A toolkit can be defined as follows:

> A collection of programming subroutine libraries that software developers can use to make programming easier. Instead of rewriting all of the code for common routines, toolkits provide pre-written routines [Ltd05].

DISCOlab should be an interface for developing shared display systems on top of the platform. The focus of the research will be on what functionality such a toolkit should offer to a developer. What properties do shared display systems share should be identified.

Shared display systems will be one frequent software that will run on the UbiCollab platform. A toolkit is therefore needed so developers do not have to implement functionality that all these shared display systems have in common.

The expected results for the project are as follows:

- A toolkit for developing shared display systems on UbiCollab, with focus on providing access to the UbiCollab services, should be developed.
- A demonstrator of a shared display system will be developed, using the toolkit. This demonstrator will evaluate how the toolkit can be used to develop a shared display system running on the UbiCollab.

1.4 Research method

The research method of this project is scenario driven and prototype based. The beginning of the project will be used to do a literature review by studying existing literature on shared displays to get an understanding of the research field. Two scenarios, in

appendix A and appendix B, are developed. Scenario "Shared display scenario" in appendix A was developed in this master thesis as a mean to uncover requirements to shared display systems. Scenario "UbiCollab scenario" was first developed by [Sch04, Bak04, Gon04] and later extended by [SJH04, BR04]. These will be used to analyse the domain of shared displays and what properties shared displays share. This analyse should end up with requirements for DISCOlab. DISCOlab will then be designed and implemented. A demonstrator will be developed using DISCOlab. The demonstrator will be used to verify that DISCOlab can be used to develop a shared display application.

1.5 Relation to previous work

This project is related to the previous work on the UbiCollab platform done by [Sch04, Bak04, Gon04, SJH04, BR04]. The aim of this work is to use the functionality in the UbiCollab platform by creating DISCOlab, for development of shared display systems on the UbiCollab platform. DISCOlab will use the platform to provide information to the users, helping them in the collaborating tasks by contextualising the information in the shared display applications using the UbiCollab platform. The platform will be used to offer the users transitions between shared display applications by offering an API for this. Some time will be spent on reviewing the previous version of the platform to improve architecture and implementation. These improvements should not remove any functionality already provided.

1.6 Restrictions and challenges

Working on this project makes me dependent of co-workers results. The project is building on previous work done on the UbiCollab platform and work done in parallel with this project. DISCOlab should offer the functionality of the UbiCollab platform. In relation to this, it is desirable that work done on the platform in this time period also is incorporated in the toolkit.

The main challenge of the project is to be open-minded enough to think new and create an architecture generic enough, but still supporting development of shared displays. The domain of shared displays is vast and the properties of shared displays can be many. Identifying good solutions is therefore a complex process.

1.7 Report outline

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

2 Introduction to shared displays: An introduction on shared displays is given. A categorisation of shared displays is done referring to shared display systems.

3 Problem elaboration: Two scenarios are analysed ending up with a list of research questions that a toolkit for shared display development should support.

4 Related work: Gives an overview of the work done on the UbiCollab platform and a shared display client running on the platform. Then requirements for toolkits supporting groupware development are presented and aspects of toolkits and creation of toolkits are presented.

5 Analysis: A list of requirements for DISCOlab will be elaborated based on the research questions in the chapter 3.

6 Conceptual design: A design of DISCOlab will be presented, defining a conceptual model. An API for DISCOlab will be given.

7 Prototype: Based on the design in chapter 6 a prototype of DISCOlab will be developed.

8 Demonstration: To demonstrate and evaluate how DISCOlab can be used to develop a shared display, a simple demonstration application will be developed and demonstrated using DISCOlab. To extend the demonstration, examples of other shared display applications will be given and a discussion on how DISCOlab would be used to implement the applications will be given.

8 Evaluation: An evaluation of the work will be done, looking into the research method and the demonstrator's ability to work as an evaluator of the prototype. Functionality in the prototype and the conceptual model will be discussed along with shortcomings and simplifications in the work.

9 Conclusion: Presents a conclusion of the project with a summary of what has been achieved. A list of future work will be also be presented.

Chapter 2

Introduction to shared displays

In this chapter I will introduce the reader to shared display systems, so that the reader can get an understanding of what kind of support DISCOlab should provide in developing shared display systems.

> Prehistoric cave drawings, framed photographs, blackboards in classrooms, posters, flip charts, road-signs and point-of-purchase displays are all visual forms of communication that play a vital role in the way we understand, navigate and behave in the environment. They offer a rich resource around which conversations and group activities are structured, complementing verbal communications and shaping group dynamics. They act as important cultural reference points in the construction of shared meanings, beliefs, desires and the memory of groups and communities [OPCR03b].

All the displays mentioned above can be categorised to be shared displays. Today more and more of these displays have been digitalised. In our daily life we do not often even notice that the displays have transformed to be digital displays. For instance, the screens at metro stations have changed from being mechanical to digital screens in many places. The advantage of this is not only that each display does not have to be maintained on-site, but also that the display can be used in different settings and for different purposes, for example in case of an emergency at the metro station. The screens can be used to give valuable information to metro customers.

I have chosen to organise the work in this chapter into three main dimensions, location of people, type of support and time. The dimensions have been chosen because these dimensions address the domain of shared displays, covering all systems. Remember that the dimensions are used analytically and that systems refered to in the text often have functionality that belongs to different dimensions.

2.1 Location of people

The location of people plays an important role in requirements to a shared display and what functionality it should provide. By location of the people it is most often refered to if people are co-located or distributed.

2.1.1 Shared by co-located people

Shared displays that is shared by co-located people can be of many types. Such can be information-screens in public spaces that provide information to customers, users, etc. Another example of shared display of such a type can be a public screen that is used for community building, e.g. the KOALA system [DF04]. Some displays for co-located people are specifically design to support collaborative tasks. Electronic whiteboards support this, where users can draw and make illustrations for colleagues. A presentation tool on a projector is also an example of a shared display used to support collaboration and information sharing. [SHH04] presents Caretta, a system that integrates personal and shared spaces to support face-to-face collaboration. Users of Caretta can discuss and negotiate with each other in the shared space by manipulating physical objects, while they individually examine their ideas in their own personal spaces. In [ASGH+99] an interactive landscape for creativity and innovation is presented, i-LAND. The environment has several digital surfaces available for co-located collaborative work, an interactive electronic wall (DynaWall), an interactive electronic table (InteracTable), and mobile and networked chairs with integrated interactive devices (CommChairs).

2.1.2 Shared by distributed people

For people that is in different locations, other shared display applications can be used. An example of such can be the Microsoft chatting tool, MSN [MSN], which include many features like shared desktop, video conference, etc. and that gives support to people in collaboration and information shared across locations. In [CF04] a system for lightweight photo sharing, particularly via mobile devices, is presented. This provides people the possibility to easy share photos from mobile devices and on desktops.

Often shared displays for distributed users provide more information about the other users of the application. This to improve the conditions for collaboration and awareness between the users of the application. This can be information of who is interacting with the application, where the users are, their presence, i.e. if they are available or how they can be contacted, user profiles, pictures or metaphors communicating the motional state the user, etc.

2.2 Type of support

Shared display systems can be said to support two types of interaction and collaboration between people, formal and informal.

2.2.1 Formal

> Formal collaboration is characterised by being intended, scheduled in advance, having an agenda [Far02].

Shared display systems that support formal collaboration are many. Maybe one of the most common shared displays that is in use in this setting today is the projector showing a presentation. Another group of applications to gives this support has been meeting tools for distributed personal, like Netmeeting [Net]. Both this tools are used having an agenda and are scheduled in advanced. Neither of them induce collaboration, but instead support it when it is needed.

In [NC005], a video conferencing system, MultiView, is presented. MultiView supports collaboration between remote groups of people. Video conferencing systems are characterised by being intended, scheduled in advance, having an agenda an are thus most often supporting formal collaboration.

2.2.2 Informal

> Informal collaboration consists of spontaneous and unplanned interactions that occur frequently and transparent within the organisations. Informal collaboration is crucial for developing working and social relations, and for long-term learning [Far02].

The shared display systems that support informal collaboration provide support to trigger interaction between people and to initiate information exchange. They also mediate awareness information between people, not presence at the same location.

In [HRS04], an instant message system for informal collaboration on large, shared displays is presented. [DF04] presents a shared display promoting informal cooperation and presence in a community of people sharing physical spaces in their office spaces. [MIEL99] presents an augmented whiteboard interface designed for informal office work, Flatland. The work support long-term, informal use of whiteboard in an individual office setting.

2.3 Time

Collaboration between people is always related to a time scale. Some collaboration must be done synchronously and some collaboration is done asynchronously. [QN001] presents an integration of a collaboration-friendly Internet protocol, WebDAV, that was used to implement a groupware system which can support document-centric asynchronous collaboration activities, e.g., collaborative document authoring, collaborative document management, etc., as well as an industrial strength product, Lotus SameTime [lot] to provide synchronous collaboration support, e.g., team awareness, instant messaging, shared whiteboard, and IP audiovideo conferencing.

2.3.1 Synchronous

Synchronous can be defined as follows:

> A type of two-way communication that occurs with virtually no time delay, allowing participants to respond in real time. Also, a system in which regularly occurring events in timed intervals are kept in step using some form of electronic clocking mechanism [Syn].

Synchronous collaboration with shared display applications is as stated above collaboration that happends in real time. Systems that support this binds peers and communicate their actions in no time delay. Such systems can be synchronised whiteboards, media spaces that synchronises video from two locations so that participants in meetings can view each other, desktop sharing where several people can view and manipulate the same file at the same time.

A typical group of synchronised shared display applications are; media spaces - computer-controllable networks of audio and video equipment used to support synchronous collaboration [Gav92]. Media spaces are applications that give small real time video captures or periodic pictures of members of a group. Through media spaces members of the group can observe what the others are doing, if they are in-office, their mood, if they are busy or available for dialog or interaction, etc. A simplification of this is also available that use avatars and virtual reality to provide the same effect [Gre96].

2.3.2 Asynchronous

Asynchronous can be defined as follows:

> A type of two-way communication that occurs with a time delay, allowing participants to respond at their own convenience. Literally not syn-

chronous, in other words, not at the same time. Example of an application of asynchronous communication is electronic bulletin board [Asy].

The asynchronous support in shared displays is needed when the communication and collaboration not is emergent and participants can respond when they are available. Such system can be a messageboard, a shared calendar system, etc. What is typical for a system like this is that an emitter of some information emits what it wishes to inform others of, and the receivers can read or get the information when they need it, sees it or have time to consume the information.

The Plasma Poster [CNLH03] and the Community Wall [Gra03] are examples of applications offering asynchronous sharing of information. They are designed to enable people to post and annotate information onto a large public display available to a community of users. The Dynamo system in [BIF+04] is a similar system. Users can drag their digital media (e.g. video clips, text files, digital images, PowerPoint slides, audio files and documents) onto the Dynamo surface, where it can be displayed, interacted with, organised, copied and left for others.

Summary

In this chapter I have explained key dimensions characterising shared display systems and presented some examples on shared display systems that gives support in different kinds of collaboration. In the further research I will keep these dimensions in mind.

Chapter 3

Problem elaboration

A toolkit for developing shared display systems is to be designed and implemented, DISCOlab. To create such a toolkit it is important that all situations where such a toolkit is to be used have been investigated. I will go through two scenarios that are interesting for this project. The scenarios are being used as a mean to uncover requirements for DISCOlab.

When developing a toolkit that should support a developer in creating shared display systems, the discussed dimensions in chapter 2 must be taken into consideration. The toolkit must provide functionality to the developer covering the dimensions that the developer is creating applications along. The dimensions have been used to generate the scenario in appendix A and to verify that the scenarios covers the domain of shared display systems. However, the dimensions are not suited to elaborate research questions around. To elaborate research questions I have chosen to organise the scenario analysis into sections regarding Discovery, advertisements and identification of entities, Information sharing and storing, Implications for public versus private displays. and Contextualising the information.

This organisation has been done because; The toolkit is to support collaboration in a ubiquitous environments. This requires it to support dynamic discovery, advertisement and that the system is able to identify entities. A shared display system is created with the mission to share information between peers. This issue must therefore be addressed. Further the devices will often be used in both private and public spaces. This superimpose requirements as giving support to a much wider range of shared display systems and increased support for flexible use of the toolkit. DISCOlab is created to be used on top of UbiCollab and will take advantage of the possibilities this offers on contextualisation of information. It is therefore important that this is investigated.

For each section, the scenarios will be discussed and related research questions will be formed based on the discussion. The research questions will form a foundation for the

analysis and the design of DISCOlab.

The first scenario was developed by [Sch04, Bak04, Gon04] having focus on collaboration and later reviewed and extended introducing concepts like location and privacy by [SJH04, BR04]. The second scenario is developed in this project, having focus on shared displays illustrating several different types of collaboration through separate shared display systems. The scenarios can be found in appendix A and appendix B.

3.1 Identifying key dimensions in the scenarios

A high level description of the scenarios will be done in the section, identifying key dimensions of shared displays in the scenarios. These key dimensions are the same as I have described in the previous chapter, chapter 2.

3.1.1 Shared display scenario

This scenario, see appendix A, has examples covering all dimensions for a shared display. Two types of shared display systems are presented in the scenario. The first is a messageboard and the second is a presentation and drawing tool.

The messageboard hanging in the office is a shared display that is used both for co-located and distributed people. The screen itself is only used by co-located people to view information on, but distributed people can send messages to it; this way the messageboard is becoming a hybrid of the two dimensions.

The messageboard shared display is mostly giving informal support for the users. The users can send messages to the messageboard that can induce informal collaboration. A messageboard is also an asynchronous shared display system. This because messages can be sent to the messageboard and viewed at a later time.

The presentation and drawing tool is a synchronous application. All information is shared real-time and can not be viewed later. The application is mainly supporting distributed collaboration. It is supporting co-located collaboration also, in the way that users in the same location can have their own screen and control it from their laptop, etc. In distributed environment users can all get the same information at the same time, only limited by the device capabilities and infrastructural limitations. The tools are giving support to formal collaboration.

3.1.2 UbiCollab scenario

In this scenario some of the dimensions are covered.

This scenario has examples of people using a shared display to collaborate on the same location as well as in distributed locations.

> Brian. She uses the client to display the slides she has prepared on the projector. The projector shows the file on the display, her representation widget lights up and she uses the PDA to remotely control the presentation. A small snapshot of the current window shows up on her display while she taps through the remote control commands. During their discussion Brian takes his turn of arguing. He accesses the remote control window on his PDA and jumps to some previous slides of the presentation.

In this example the shared display is used for co-located collaboration. The same screen is used by several participants and is also controlled by all of them. The following example covers distributed collaboration. Here, two screens are synchronised and the screens can be controlled from both locations.

> Checking his available devices, she can see that he has a display application similar to the one in the projector. She says to Steve that she'll synchronise their displays, so that he can also see the current slide. She selects the two devices in Ubi- Client, and synchronises them. Steve sees the slide showing up on his screen.

In both examples the collaboration is formal. It also is synchronous. Thus, the informal type of collaboration and the asynchronous time aspect are not covered by the scenario.

	Shared display scenario	UbiCollab scenario
Co-located	X	X
Distributed	X	X
Formal	X	X
Informal	X	
Synchronous	X	X
Asynchronous	X	

Figure 3.1: Dimensions covered by the scenarios

3.2 Discovery, advertisements and identification of entities

A shared display system has entities that share and display information. How these entities should be discovered by the system and how these entities advertise themselves must be addressed.

> Sylvia notices that the meeting room is equipped with a projector. She searches for the device with the UbiClient, the client comes up with the closest alternatives and she adds the projector to the meeting. She activates the projector, and it lights up and displays a welcome-screen, and a representation of her and Brian.

Having a shared display system running in a ubiquitous environment makes it important that entities can be discovered dynamically as devices move through the environment. Some kind of support for dynamic discovery must therefore be supported. The attributes used in such a discovery mechanism must also be carefully considered so that the entities can be searched for in a sensible way. For instance, should there be possible to search directly for projectors by location or should you search for devices capable of showing the content you want to share and display? The discovering process address two aspects of the system. One is related to the entity sharing some content. Here it should be detectable what kind of sharing is done. The other is the discovery of visualising entities. Related to that are questions like, what kind of capabilities do the visualising entity have and can it visualise what the sharing entity shares.

> Pepe is a Colombian consultant the company has hired for this project. He is working hand on with the building of the services the company is doing in Colombia. Before the weekend he downloaded the presentations that the others had added to the meeting. This way he can follow the presentations without having his laptop connected to the platform this morning. As the presentations proceed he can see at his mobile phone what pages and documents that are been displayed and discussed. Pepe is moving back and forth in the presentations and listens while the others speak.

This is an example of a presentation tool that shares information in several ways. In addition of having a visual view of the actual synchronised presentation, a synchronised presentation event queue is offered. The actual presentation is presentable for devices with displaying capabilities, but Pepe's mobile phone does not have the necessary capabilities. The issue here is how to discover what capabilities display devices have. A decision on automatically offer the suitable services or not, based on the device capabilities must be taken.

An entity that shares some content must relate to the fact that in a ubiquitous environment there are many types of devices with different displaying capabilities. The sharing entity should therefore offer the content that it shares in a way that makes it possible for devices to select what content or part of content they wish to visualise. Content that is to be shared should be defined in some kind of entity, for example a service with a description.The sharing entity should therefore have some definition of what services it offers and some description.
With different kinds of devices running in the system, it is probable that the devices are running on different platforms. DISCOlab must consider this when choice on technical solutions is done.

A shared displays system will consist of several entities. At least an entity that is responsible for the sharing of information is needed. In addition an entity that visualise the content must be present. These two entities can be running in the same host. Bakkevoll [Bak04] also defines an interaction client, see section 4.2, that is used to control a client visualising content. However, each entity defined in the system must have clearly defined roles of responsibilities. It is important to identify a sensible structure of how the entities in the system should collaborate and the responsibilities and authorities in such a collaboration between entities.

This allows us to formulate the following research questions related to discovery, advertisements and identification of entities:

A-1: What entities do we have in the shared display system?

A-2: How letting a shared display present information tailored to different types of devices?

A-3: How should the sharing entity communicate with several types of devices possibly running on different platforms?

A-4: How should the content that is to be visualised be associated with the entity doing the displaying of the content?

A-5: What kind of discovery mechanism would be most sensible? (E.g. Search for closest projector or for closest device capable of displaying the content that is to be shared?)

A-6: How to discover what capabilities display devices have?

A-7: Should the appropriate services for visualising content be automatically selected based on device capabilities?

3.3 Information sharing and storing

In this section an analysis of the scenarios will be done with respect to how information should be shared and stored in a shared display system.

> Sylvia notices that the meeting room is equipped with a projector. She searches for the device with the UbiClient, the client comes up with the closest alternatives and she adds the projector to the meeting. She activates the projector, and it lights up and displays a welcome-screen, and a representation of her and Brian. She uses the client to display the slides she has prepared on the projector. The projector shows the file on the display, her representation widget lights up and she uses the PDA to remotely control the presentation. A small snapshot of the current window shows up on her display while she taps through the remote control commands.

This scenario part illustrates several interesting ideas that a shared display system must consider. The first thing is the relation of the content that should be displayed by a display and the display itself, here the projector. The content, here a presentation file, is added to the projector by the UbiClient. How should the content that is to be displayed be associated with the entity doing the displaying of the content, must be considered. Another issue is where the content in the system should be stored and how should it be sent to the entity displaying the content. What entity in the system should save or keep the content and be responsible for the sharing must be addressed.

Using the PDA to control the presentation makes the PDA to be a remote control for the projector. Hence, the presentation tool running on the projector is both controllable by a PDA and is displaying content on another device. This makes it necessary to let the presentation tool to be communicating with several devices at the same time. This implicates that the presentation tool has several services for different devices. In this example the PDA is used to control the presentation and the projector to display it.

In the UbiClient Sylvia gets a snapshot of the presentation running on the projector. To do this, both displays, the UbiClient and the projector, should synchronise the content. How the synchronisation should be done and what entity that should do the synchronisation must be considered.

> Brian uses his UbiClient to invite Steve to the meeting. At the time, Steve is in the company cafeteria, logged on with his PDA. A message pops up on his screen, asking him to join the meeting with Sylvia and Brian.

Getting information on members of a collaboration instance is addressed in this example. Such information is provided by the UbiCollab platform. DISCOlab should provide this information to a shared display. The communication with the platform should be straight forward. The platform offers ways, following established standards of communication over the network. A review of how this can be done can be found in section 4.1.2.

> This Monday morning John wakes up half an our late and sends a message to the job that he is running late and that the meeting has to be postponed an hour, to 9.00 am.
>
> When Sylvia gets to the office she can see Johns message at the office's whiteboard, that runs a messageboard application.

Sending a message to the whiteboard in the office is an example of asynchronous use of the shared display. Information can be shared both in asynchronous and in synchronous ways. This kind of support sets different requirements to a system. For asynchronous sharing, some persistent data store should be present that keeps the information if the entity keeping the information shuts down. The entity doing the sharing of asynchronous information must also be running even if no clients are running.

This because an asynchronous sharing entity must handle information that might not be accessed at a specific time instance, but in some future moment.

In situations where synchronous sharing is to be done other requirements must be fulfilled. Requirements to response time and throughput of the system must be addressed. Synchronous sharing can be very load intensive on the entity doing the sharing. Risk for bottlenecks reducing the performance of such a system gets high. Architectural aspects such as centralised versus peer-to-peer must be considered in systems for synchronised sharing to reduce risk of bottlenecks, but also for making it easier for impulsive sharing of content at a peer.

> She uses the presentation mode of the big whiteboard hanging on the wall. Halfway through the presentation John has a question and asks Sylvia if she him going one slide back. John uses his laptop to control the presentation and points at some figures in the slide asking Sylvia. Sylvia redirects the question to Mary at the other side of the table. Mary was the one designing the figures that John found questionable. Mary uses her laptop and switches the shared display to drawing mode. The display is synchronised with the other computers throughout the session. She maximises the screen so everyone can have a good look and she starts explaining while she draws.

Several aspects are covered by this example. First is how the sharing of content should be initiated and given access to by other people. In a shared display system many entities can be present that are subscribing to some content being shared. These entities must have some configuration information associated to them. E.g. a client connected to a synchronisation server need to have some preference settings and subscribing information along with what capabilities it has. It also needs to be recorded what access rights it has. What entity that should have responsibility of storing such information and how the information is to be stored must be addressed.

When sharing some content, there can be many situations where a sharing entity do not want to let others take control of the sharing. This can for example be in a presentation. Some mechanisms should be present to restrict people from taking the control. This might not be at question on the toolkit level, but in the application layer. Another issue related to protection is to have the possibilities to decide what parties should have access to the information you are sharing and what kind of access it should be. Sometimes a sharing party only want to give read access to a drawing session on a whiteboard and most of the time such a session only should be visible for a limited group. DISCOlab should have ways of setting access rights to the content being shared.

This allows me to formulate the following research questions in relation to information sharing and storing:

B-1: Where in the system should the content be saved?

B-2: Where and how should configurations be saved?

B-3: What entity should be doing the synchronising of content?

B-4: How to route and handle the information sent to a shared display system?

B-5: How should the content be sent to the entity visualising the content?

B-6: How should sharing of content be initiated?

B-7: How to support asynchronous and synchronous information sharing?

B-8: How to keep asynchronous information in the system?

B-9: How should the synchronising be done?

B-10: How to handle several people trying to control the same application?

B-11: How to protect the content that is being shared or protect the way it is being shared, e.g. restrict others from taking control of a presentation?

B-12: How to communicate with the platform?

B-13: Should the architecture be peer-to-peer or centralised?

3.4 Implications for public versus private displays

Having a shared display for public use is not the same as having it for private use. Public displays can have a big and diverse user group with demands for different functionality in opposition to a private display. This raises different requirements for public versus private displays.

> Steve opens an audio connection to the meeting room and Brian fills him in on their problems. Steve then wants to look at the slides and searches for a shared display using his UbiCollab client. The system finds a display in the cafeteria but it is unavailable for him. He then decides to startup his laptop to use its shared display capability.

This is an interesting example of sharing a presentation with a collaborating party. How should Steve access the presentation in the example? Should a person install the required application to show a presentation prior to a presentation or should the required applications be fetched from a location when needed?

A shared display used by the same user will probably be used by a few applications. With this type of use there will not be a problem to install the applications that are needed to visualise some content that is being shared. In a public room however, a shared display will probably be used for visualising multiple types of content. This

because the user groups in public rooms have different needs. In a ubiquitous environment many users will have the need of visualising some content on a public screen. The screen should have possibilities to do this regardless of not having visualised that type of content before.

Another issue that must be considered is how people should access information on public shared displays. Should people be able to give input to these kinds of displays, should they log in to the display and should it be possible that these displays can be controlled from remote locations? There probably should be ways of controlling the content sharing on public shared displays. This should be done both locally and remotely, depending on the type of content sharing.

A public display like the one hanging in the cafeteria have potentially many users with different requirements for the display. How should a sensible architecture support this type of use. Public shared displays should have ways of displaying unfamiliar content in any time at any place. This is a major issue that needs to be addressed in the design of DISCOlab.

This allows me to formulate the following research questions in relation to implications for public versus private displays:

C-1: Should a person install a the required application to show a presentation prior to a presentation or should the required applications be fetched from a location when needed?

C-2: How should information be accessed at public displays?

C-3: How should a public display support visualising different content to a user group with different demands?

3.5 Contextualising the information

The shared display system should run on UbiCollab and should in respect to that have en effective way to contextualise the information shared in the system. Below, an example from the scenario “Shared display scenario” illustrates this:

> Mark also has been seeing through presence information and context information displayed in the application which colleges that have been active and contributing to the meeting. He can see that all the others, except Pepe, are gathered at the main office in Oslo.

DISCOlab should take use of the services that are offered by UbiCollab. The information that can be associated to material being shared by a shared display application, can be collaboration instance information, resource information, privacy information filtering the content being shared, etc. The services that run on UbiCollab for this should be

accessible from DISCOlab, so a developer can use the services together with DISCOlab to develop shared display systems. An example of using the privacy services can be to use the privacy service to automatically refuse visualising of sensitive information on public displays.

When a user wants to share some content with the collaboration instance he is a member of, this should be easy to do. Another example using the services at UbiCollab is; a user that subscribing to some content should have the possibility to get the content sent to the nearest display the user has available. An API for a shared display should provide this functionality to a user.

This allows me to formulate the following research questions:

D-1: How should shared content be associated with collaboration information?

D-2: How should the basic services of UbiCollab be used to give contextual information?

D-3: How should DISCOlab provide easy access to the shared content in a contextualised way?

Summary

The scenario analysis has raised a set of questions and discussions around these. These questions and discussions form a basis for the analysis and the design of DISCOlab. It is important that questions in this chapter are considered when analysis of the functionlity and design of DISCOlab is done, so that the result of this project is a toolkit that has the needed functionality.

Chapter 4

Related work

In this chapter I will first present the previous work done on the UbiCollab platform. This work is highly relevant for my project because DISCOlab should use the platform in the development of a shared display systems. After a review of the work done on the platform, I will present some related work in the field of toolkit development.

4.1 UbiCollab platform

The platform was developed by [Sch04] the spring of 2004. The work done by [Sch04] builds on a previous version of the platform as reported in [DFS04]. UbiCollab provides support for collaboration in a ubiquitous environment. It does however not have the responsibility of providing the actual services for specific type or domain of collaboration (like instant messaging, cooperative programming etc.). That is, the platform enables support for usage and management of such services, but does not provide such specific collaborative services itself.

UbiCollab consist of several native platform services, see figure 4.1. These provide the support for collaboration in a ubiquitous computing environment. By keeping the platform clean like this, the generality of the platform is ensured and the focus can be on developing new kinds of collaboration support instead of reinventing existing collaborative services. Adding new kinds of collaboration support is done by creating new services that are plugged into the platform.

To represent the abstract collaboration concept, Schwarz uses a collaboration instance. This is an entity which is meant to capture a real world activity, or context, of collaboration between people and resources they use. A person entity represents a human participant while a resource can be a physical device or electronic information. By creating a conceptual model centred around the collaboration instance several issues can be solved fairly easy. This gives a focus point for all the services and tie the whole platform together as a coherent system. Another important aspect with this solution

is that it enables resources and other entities to be connected to several collaboration instances. It also enables collaboration instances to be aggregated with others and still maintain their components.

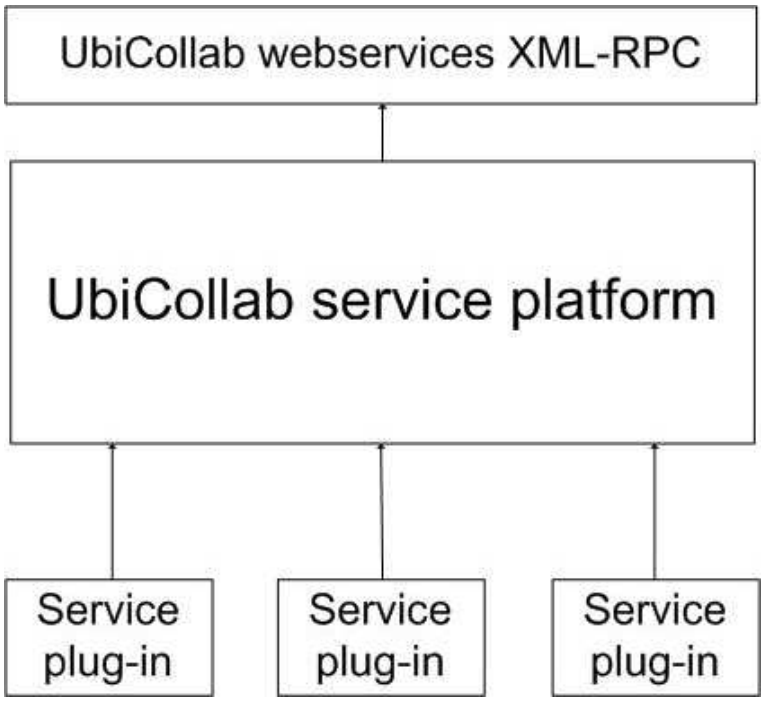

Figure 4.1: Platform overview

4.1.1 Platform architecture

The platform consist of modular services that cooperate with each other but run independently, see figure 4.2. The most complex and advanced service is the collaboration server which provides the main functionality that enables collaboration between clients. The directory service is another important service. It provides basic resource management and is used to provide information on which resources are available at a given time. It collects information on which resources are available from resource collectors which run on the clients. This is a good solution since it enables dynamic adding and removing of resources as the device collectors discover or lose devices.

The presence service saves presence values to a user. A user's presence value will be set to some kind of context. A context can be a collaboration instance or a service used for collaboration. This separation of user presence values makes the presence information more accurate.

When it comes to the location service and the privacy service these are under development. Developing the location service was started the fall 2004 by [SJH04] and is continued by Setså Jensen [SJ05]. The location service has the responsibility of finding positions and locations of users and resources in the environment. A position is

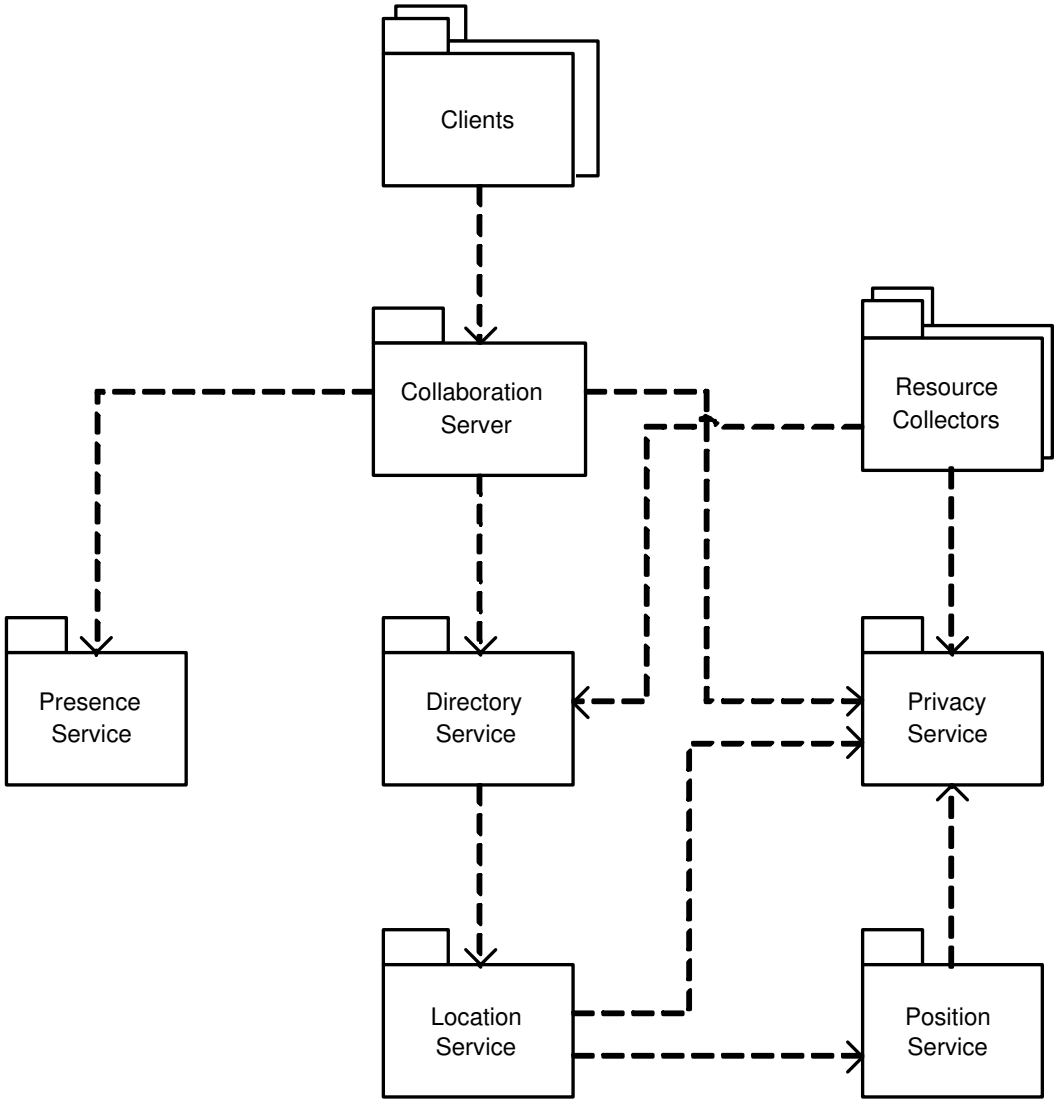

Figure 4.2: Platform architecture

here refered to be a point in space, while location is some place in space connected to a context. The location service should give answers to where resources and users are in space. It should associate this information to the different contexts the users and resources are in.

Braathen and Rasmussen started to work on the privacy service the fall 2004 [BR04] and have continued their work the spring of 2005 [BR05]. Privacy is an important issue in ubiquitous computing because the system is everywhere and keeps monitoring the user. The privacy service is a central service that keeps privacy profiles on users. It will work as a proxy for the other services, so that the system can be sure that the requester is an authorised entity. The collaboration server is an authorised service, but before distributing any information to a requester it will have to go through the privacy service.

4.1.2 Communicating with the platform

Internally the platform is communicating between the native services over SOAP [SS00, SOA]. Each service is a UPnP enabled service running independently of the other services in the platform. UPnP is short for Universal Plug 'n Play and offers pervasive peer-to-peer network connectivity of PCs of all form factors, intelligent appliances, and wireless devices, by supporting dynamic discovery of devices [UPn].

On the UbiCollab server it is defined an API. This API makes it possible for clients to communicate with the server over webservices. The communication is done over SOAP. DISCOlab will use this API to access the services the platform has. The API can be found in appendix C.

4.2 KOALA - A shared display system

KOALA is a shared display system that lets you use different kinds of devices to display PDF presentations. It was developed by [Bak04] the spring 2004.
KOALA is relevant to this project because it is an example of a system DISCOlab is to provide support to develop. I will in this section make a review of the KOALA architecture.

Architecture

Bakkevoll defines three entities in his design. A shared display client, a shared display server and an interaction client. These can be seen in figure 4.3.

The shared display client is the entity visualising the content to a user. It is running on a display device. The shared display client can receive inputs from a user. It is connected to the UbiCollab platform and has access to information in it. The shared display client is visible to the platform.

Synchronisation of content is done by the shared display synchronisation server. This is a server only visible to shared display clients. Shared displays communicate via the shared display server.

A last entity in the system is an interaction client. This is a client that can control the a shared display client. The interaction client sends commands to a shared display client, which execute the command and forwards the change to the shared display server. An interaction client also has access to the UbiCollab platform and is visible to the platform.

The content sharing in KOALA is done in a centralised fashion. Bakkevoll made this decision because the UbiCollab platform had a centralised architecture. Further he

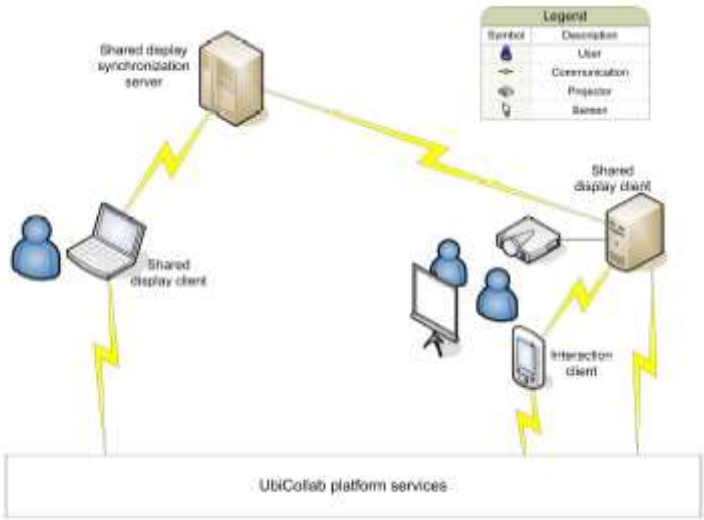

Figure 4.3: Architecture of KOALA

assumed that this approach made the forwarding of updates easier for the clients. This because various clients tends to have different interests of updates and the filtering of messages would be better to have on a centralised server than in each device. He emphasises that the shared display clients should be kept simple, not to make resource constrained devices overloaded.

The communication with the platform and the interaction clients is done over SOAP. The KOALA display is UPnP enabled. This makes the display discoverable by the platform and interaction clients that is searching for a display to show a presentation at.

4.2.1 Using DISCOlab to develop KOALA

When developing KOALA having a toolkit like DISCOlab would be beneficial. Bakkevoll has in the evaluation pointed at some weaknesses with KOALA and I will in the following point out that these weakness might been avoided having a toolkit available.

> Yet there are several problems with the design. If a shared display client is made that supports more sophisticated functionality, like annotating or editing a document, problems will arise. First, there is a problem when a shared display client in use is synchronised with another shared display client. The client would have to send the whole state, as the server does not have a representation of the state. Furthermore, a system with locking the server to make updates would have to be made, otherwise the clients would end up with inconsistent states.

In his work, Bakkevoll designed a shared display system capable of just supporting what it was do support. With that design future improvements and extensions on functionality will be difficult as pointed out by Bakkevoll. If a toolkit was used to develop the KOALA system, Bakkevoll would not have to worry about such low level implementation burdens, as those should be handled by the toolkit.

> KOALA displays information about who is controlling it in the lower right corner. Only the last action is displayed here, and this is a weakness. The text may change while the a user is looking the other way, so the information is easy to miss. At the same time, it contains no information about when the action was performed. A more sophisticated approach would be to display more than one action, and perhaps use fading and colour to give a visual cue about how much time has passed since the action was performed. This is left as future work.

The KOALA system has received much attention in infrastructural work and less in the design and giving support for collaboration through visual queues, etc. A developer that is to create groupware and that do not have tools for doing this, is often spending the energy on infrastructural problems, low-level implementation and distributed data processing [Gre04]. If this already was provided by a toolkit, Bakkevoll could have used his resources on providing more functionality to KOALA that would ease and enhance the collaboration for users of KOALA.

4.3 Toolkit properties

Much work is done on developing collaborative groupware. However, much of this work has been consisting in re-inventing low level implementation solutions. Creating groupware is an complex task, a task getting even more complex every time low level functionality has to be developed.
Toolkits for developing groupware should therefore be developed. Greenberg [Gre04] have some arguments for what such a groupware toolkit should be characterised by:

- Be embedded within a familiar platform and language in common use so that people can leverage their existing knowledge and skills.
- Remove low level implementation burdens common to all groupware platforms (e.g., communications, data sharing, concurrency control, session management).
- Minimise housekeeping and other non-essential tasks.
- Encapsulate successful design concepts known by the research community into programmable objects so they can be included with little implementation effort.

- Present itself through a concise API that encourages how people should think about groupware.
- Make simple things achievable in a few lines of code, and complex things possible.

[Gre04]

Further on Greenberg presents some common elements to real time distributed groupware applications. He identified these elements that should be available to programmers:

- A run-time architecture automatically managed processes, their interconnections, and communications; thus programmers did not have to do any process or communications management. This came for free.
- Session managers let end-users create, join, leave and manage meetings. A selection of session managers came as pre-packaged interfaces, and the programmer could use these "out of the box". However, the programmer could craft their own session manager if they wished.
- A small set of groupware programming abstractions let a programmer manage all distributed interactions. Through an RPC-like mechanism, the programmer could broadcast interaction events to selected participants. Alternatively, the programmer could manage interaction via a shared data model: programmers would then think about groupware as a distributed model-view-controller system. Local user actions would change data in the shared model, and remote processes would detect these and use the altered data to generate the view.
- Finally, groupware widgets were included that let programmers add generic groupware constructs of value to conference participants. Our first widgets were telepointers, which a programmer could add with a single line of code. Later widgets included awareness widgets such multi-user scrollbars and radar views.

[Gre04]

Creating toolkits is an advanced process. The toolkit designer does not write applications, but writes software to be used in applications; the details of those applications, what they will do, how they will do it, and who will use them, are not available at the time when the toolkit is being written [DE00].

Dourish and Adwards [DE00] define aspects of infrastructural flexibility in relation with toolkits development. Data distribution concerns the ways in which the computational representations of data are available to participants in the system. Consistency

control handles access to distributed data to maintain consistency in each users view. Access to the data store may be governed by permission settings, by roles, or by opportunity. Sessions might arise through the individual actions of collaborators, or may be set up explicitly, and might be managed on an invitation basis, or regulated by an individual.

In [Dou95], Dourish outlines three aspects of flexibility in the design of CSCW systems. Static flexibility refers to the extent to which the system can support a variety of collaborative needs (such as synchronous or asynchronous work). Dynamic flexibility reflects a systems ability to respond to the changing circumstances of a collaborative session (such as changes in group membership). Implementation flexibility refers to the ability of a system to operate over a range of implementation substrates (such as different forms of network or network topology, or different approaches to data store management).

Software toolkits ease the implementation of software systems by providing reusable components and behaviours designed to be applicable in a range of circumstances. As well as reducing effort and speeding application development, this reuse of software components also offers a common conceptual framework for application development, which can aid both application designers and users. The components themselves arise out of common patterns of software structure that occur across a range of applications in a particular domain. objects. Applications are constructed using these components as building blocks [DE00].

4.4 Towards a toolkit for creation of shared display systems

The publication "Supporting Extensible Public Display Systems with Speakeasy" [BEN+03] presents, Speakeasy, a framework for supporting extensible public displays. Speakeasy is characterised by the following two characteristics:

- Supports construction of applications that can use new devices, services and networks, without requiring any a priori specialised knowledge.
- Supports indirectly collaborative use of public display systems by permitting use of public displays over geographically distributed participants.

It is pointed out that a Extensible Public Display System needs to fulfill the following requirements:

1. It needs to be able to display virtually arbitrary content. It is pointed out that this is an unworkable approach because new content types is appearing all the time.
2. In order to interact with devices and services on the network, public displays must be able to dynamically discover them as they appear. Thus, the public display must support a wide range of discovery protocols.

3. A public display must not only be used to view and manipulate media and content, but also control devices and services. Thus, the controls for such services and devices must be integrated into a display's framework.

All of the above enumerated requirements represents different dimensions of dynamically extensibility: the ability for a display system to acquire new functionality not specifically built on creation time. Such extensibility is crucial for supporting evolution of the system: the ability to adapt to change - such as the presence of new devices.

The publication further presents two ways to achieve such dynamically extensibility. One way is to have an centralised approach where new versions of the display system is dropped supporting new protocols, device types, data types and so forth.

The other approach is decentralised. Speakeasy takes this approach and has a peer-to-peer "marketplace" of services. Rather than relaying upon the original developers to establish standards of compatibility to devices and services, Speakeasy requires the service or device to provide the new data type or protocol behaviour to the display from the device itself.

Speakeasy defines a small, fixed set of meta-interfaces. The idea is that rather than attempting to define some large collection of standards for how every foreseeable device or service should inter-operate, the meta-interfaces defines the ways in which components acquire new behaviour. These meta-interfaces are as follows:

- Data transfer: used to extend clients to new data transfer protocols and new media handling abilities. A data transfer is realised setting up a source and a sink. The source sends an endpoint object to the sink. The endpoint object enables the sink to reproduce the data send from the source, see fig 4.4.
- Aggregation: used to allow clients to acquire new discovery protocols and operate over new network transports. The aggregation is done through aggregated components that contain other components, e.g. file systems, discovery protocols which appear to contain all of the components discoverable through that protocol, etc. Further aggregates can act as "bridges" to new types of networks, fig 4.5.
- Meta-data: used to allow clients to query the capabilities and attributes of components on the network. The information is organised in simple key-value pairs, with no special organisation imposed on either the keys or the values. The meta-data is primarily intended for human consumption.
- Control: used to allow components to deliver new, custom user interfaces for controlling them to applications and users.

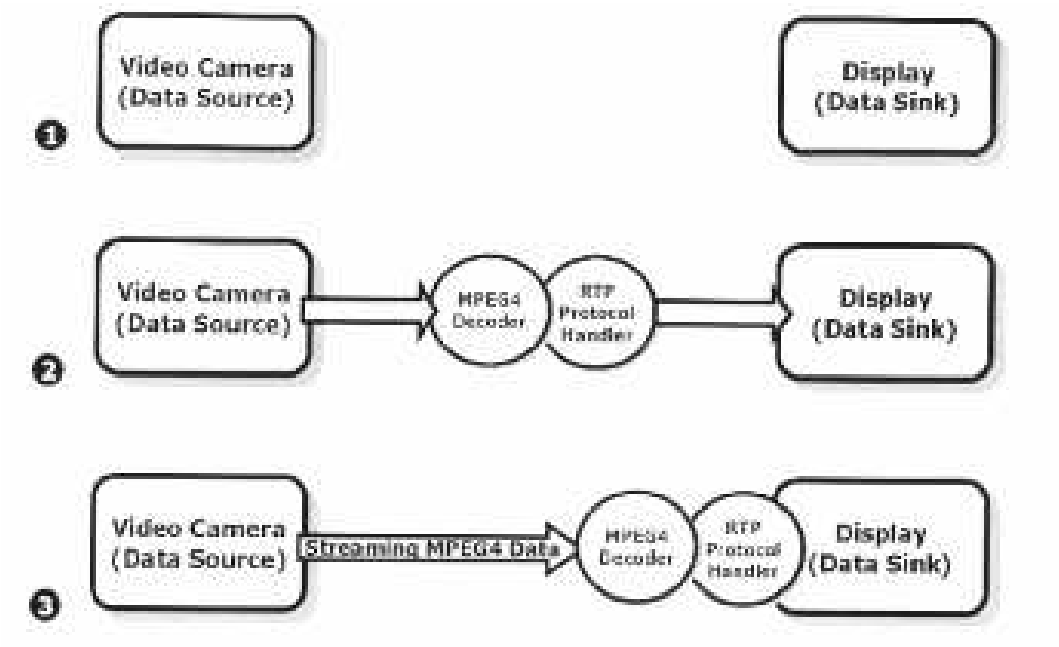

Figure 4.4: Speakeasy data transfer

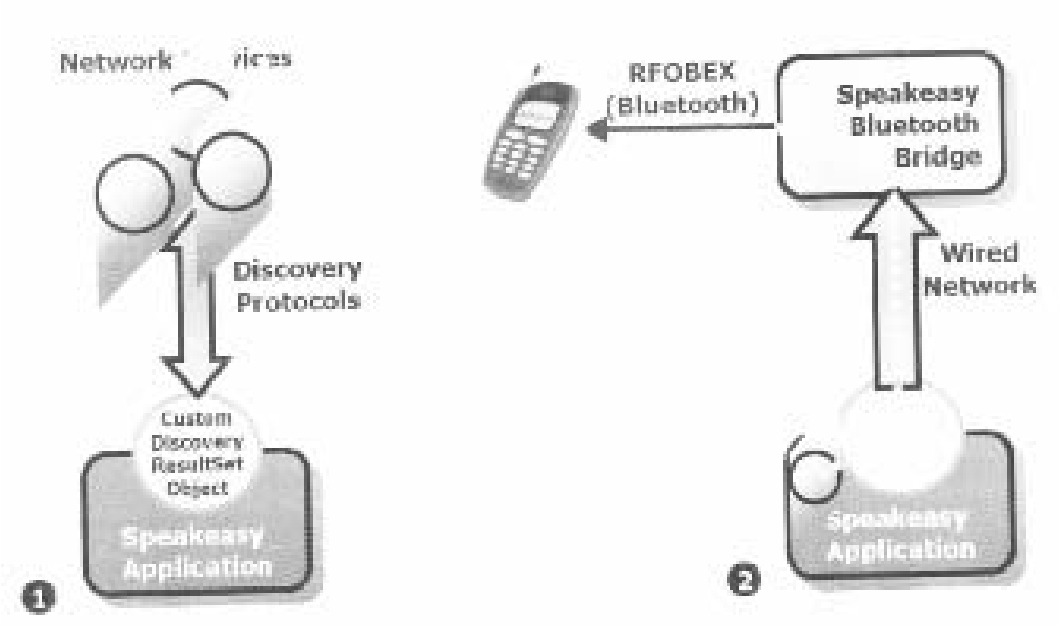

Figure 4.5: Speakeasy aggregation

Speakeasy is moving towards being a toolkit for shared displays. It address requirements like dynamically extensibility and creates a flexible framework for shared display systems. However, Speakeasy supports creation of standalone systems and no collaborative platform for contextualisation of information is provided.

Summary

In this chapter we have looked into the UbiCollab platform. The architecture of UbiCollab has been described and important concepts have been presented. We have looked into how the focus could have been on providing better collaborative support in KOALA if a toolkit had been used to develop the system. Finally, important properties of toolkits have been presented.

Chapter 5

Analysis

Based on the research questions in chapter 3 and the related work in chapter 4, entities in a shared display systems will be defined and a set of requirements will be outlined.

5.1 Defining system entities

This section will address research questions A-1, A-2, B-3 and B-9.

A shared display system consists of several entities with different tasks. Each entity should have clearly defined tasks and responsibilities. These entities must be identified and defined by DISCOlab, so that a developer can take them in use without difficulties.

The use of displays always demands an entity that does the visualisation of information, a visualising display. It is also needed an entity that does the sharing of the information, let us call this a content server. This can for instance be a synchronising server for a real-time sharing application or a message server for an asynchronous sharing application. Thus, a content server should handle tasks like synchronisation of content. The way the synchronisation is done is left to the developer using DISCOlab. This because the synchronisation is dependent on the application to be developed. In addition some ability to control the sharing must be present. The KOALA architecture defined an interaction client for this.

It is important to separate between the concept of an entity and a physical device. One physical device can have several entities running or the entities in the sharing display system can be running on different devices. This depends on the nature of the application. For instance, an application for presentation sharing can have the sharing entity on the client with a visualising entity running in the same client. The sharing entity can at the same time share information to other visualising entities running on other clients. The presentation can have an interaction client - a control panel, that sends commands to the content sharing entity.

An interaction client is not necessarily very different from a visualising entity. Both entities have a software client that interact with the sharing entity. The interaction client sends information to the sharing entity and the visualising entity receives information. Both entities can use the same protocols to exchange information and establish contact with a sharing entity. There are however some differences that makes it natural to let DISCOlab define both entities. An interaction client have aspects like protection and access control to the content being shared. This will be discussed more in section 5.5.

To let visualising and controlling entities easily be separated in the system, let devices with different capabilities and users with different desires be supported, a sharing entity should divide the sharing and controlling of the sharing entity into reasonable tasks. These tasks can be handled by entities called services. A service can for instance be a control service that handles the control of a presentation. Another service would be the presentation itself. The third service defined could be an event log for what is happening in the presentation. This way an entity with access-rights and capabilities can subscribe to all services. A limited device, i.e. a mobile phone, would maybe only subscribe to the event log. While a public monitor without input interface would only subscribe to the presentation and the event log. The visualising entity and interaction client should be defined by such services. The service would define ways of exchanging the information to be shared, how to control a sharing session, handle access control and protection, etc.

Another entity that could be present in the system could be an application library having downloadable applications, see figure 5.2. From the library an entity can fetch an application on demand and run it locally. This will make it possible to search in the system for applications capable of sharing, visualising or controlling specific file formats or media formats. If the supporting application exists an entity can display it on the fly without any previous configuration. Another advantage is that a system manager do not have to install applications or updates on each client, but can limit it to one or a few instances. This makes maintenance easy. Another option is to let the application be installed on the entity doing the sharing of some content. An entity wishing to visualise or control some content can get the application to visualise or control the content from the sharing entity directly. This will work in an architecture where sharing entities will be running independent of visualising clients. In an architecture that supports the initiation of sharing in a more add-hoc fashion with a peer-to-peer sharing of content, the system either has to have all applications installed into the sharing entity or a well known library that the system can get applications to do the sharing, visualising and controlling of some content sharing.

Thus, the most flexible solution will be to have sharing entities that have interfaces to the visualising and interaction entities and that the sharing entity can connect to an application library if it does not have the applications to do the sharing. However, this solution will be very complex and to ambitious for this master thesis. Instead the more simple but still powerful solution having sharing entities with the needed applications already installed should be chosen, as in figure 5.1.

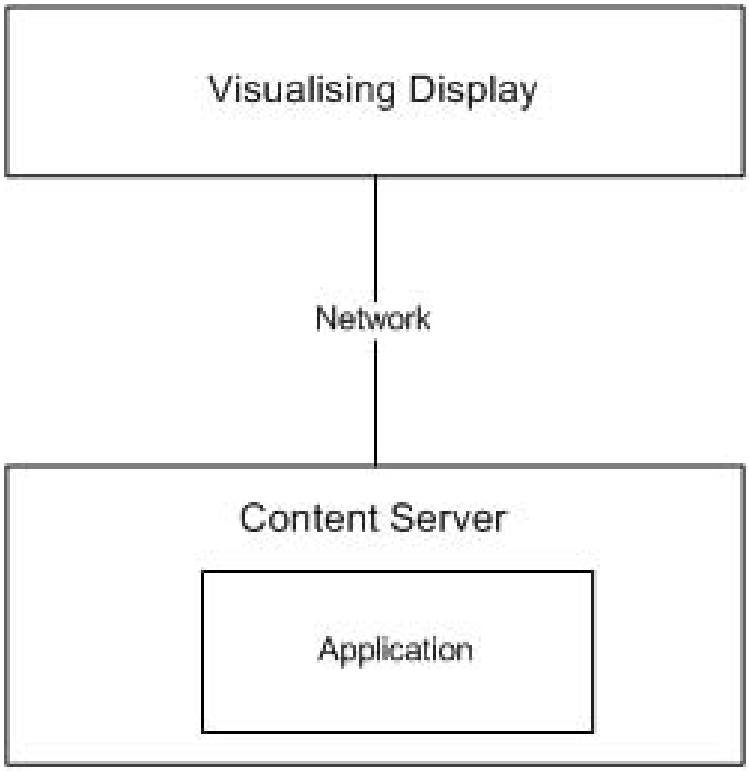

Figure 5.1: Architecture without application library

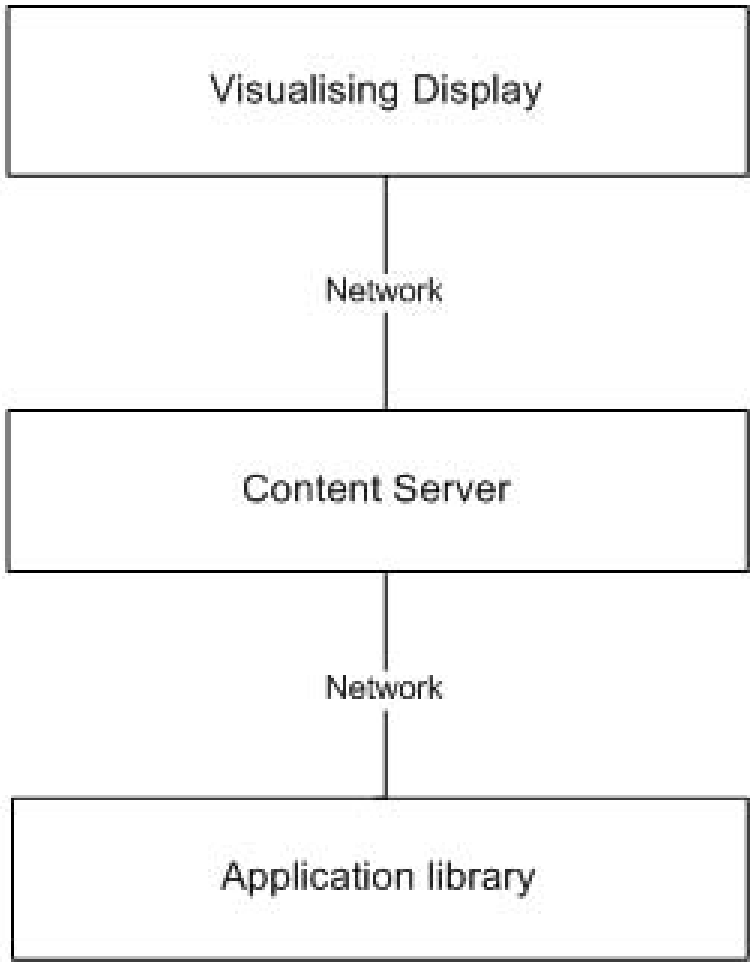

Figure 5.2: Architecture with application library

Requirements

DISCOlab should define the following entities:

R-1: Content Server: Does the sharing of some content. The Content Server is defining one or more services that the content sharing and controlling are divided into. The Content Server can run on the same host that is doing the visualisation of content or on a separate host. It should do the synchronisation of the content.

R-2: Service: A Service is defined by the Content Server and thus promoted and managed by the Content Server. A Service should be a program that visualise or controls sharing of some content. The Service should also define methods that makes it possible to interact or control a service without running the program defined by the Service. Services can be separated either into Services for control, for sharing and an event log or for device types.

R-3: Visualising Display: The visualisation should be done by a display. The visualising display should be part of a client that runs on a device. The Visualising Display should define a content-pane on which it can display Services. This content-pane should be used both to visualise content and to control the content being visualised. A developer implements a GUI-client and adds the Visualising Display to the client. The developer can decide size and other attributes of the Visualising Display depending of the device it should run on.

The tasks description of the entities is preliminary. More requirements will be given to each entity during the analysis.

5.2 Discovery of *Content Servers* and *Visualising Displays*

In this section research question A-5, A-6, A-7 are addressed.

Since the toolkit is to be used in an ubiquitous environment, discovery of the system entities must be done dynamically. The Content Servers and Visualising Displays must advertise their presence when they are running, so that a user who wants to use either a Visualising Display or a Content Server for a specific task can find it easily. The Visualising Display and Content Server must be discovered through the UbiCollab platform. Visualising Displays and Content Servers with different attributes describing the capabilities they have, resolution, file format support, media support, location, available etc. It should be possible to include criterias based on attributes that describe such an entity. A set of configuration attributes should be defined in the entities making them searchable.

For a user or a device it should be possible to subscribe to a Content Server or one or more of its Services. This makes it possible for an user to subscribe to a Content

Server and always get the shared information from the Content Server independent of what device the user is working on. The user should then get the Services that the device is capable of displaying excluding any Services the user explicitly has refused to subscribe to. A device should get access to the Services it is capable of visualising. The decision on what Services the device is capable of visualising is upon the Visualising Display to decide. The decision will be based on the service description submitted from the Content Server to the Visualising Display. The issue in relation to discovery, is how this subscribing information should be stored in the system? For the device the solution is straight forward. The device can choose what content it should subscribe to and un-subscribe to for each session. Remotely this can be done the same way, by letting the device take the initiative to contact the Content Server. For the user this is somehow more complex. A user can choose to subscribe and un-subscribe to a Content Server in one session, but some subscribing might be interesting for the user to have a longer duration, for instance subscription to a messageboard for the project. When the user subscribe and un-subscribe in the same session there is no problems. It is for the longer lasting sessions that the subscribing information must be stored and accessed in a sensible way. Different alternatives are available. A user can have a configuration file on the UbiCollab platform that among other things have subscribing information. This solution however makes it necessary to alter the platform a bit and add some functionality to it. The positive thing with this solution is that it is easy to find such information for users and it does not load the system more than necessary. A configuration file can also be used by other future applications. Another solution is to have a subscriber list on each of the Content Servers. On login a client with a Visualising Display will automatically search through the available Content Servers to see if the user is subscribing to any of them. This solutions omits altering the platform, but makes the searching for subscribing information much more bothersome.

Due to time restrictions the feature of subscribing to Services will be left for future versions of the toolkit. The solution will complicate the work of developing a first version of the toolkit, while a subscribing feature will not be highly dependent of how the toolkit is designed in this version.

Requirements

R-4: The Content Server and Visualising Display should be dynamically discoverable by the UbiCollab server. Thus, they must run software that is supported by the platform for this use.

R-5: The Content Server and Visualising Display must have a description list or a set of methods that reveals information about the entity. This list must be formed in a way that makes it searchable.

R-6: A Visualising Display shall decide automatically what Service it will subscribe to.

R-7: Subscribing to Services will only be done in one session. The possibility for a user or device to subscribe to Services over several sessions is left as future

work.

5.3 Information storing and sharing

This section is answering the research questions A-3, B-1, B-2, B-4 and B-12.

In a shared display system, information is to be shared. The sharing can either happen in a distributed or a co-located way. In distributed sharing the sharing is done on multiple hosts with one or more spectators on each host, while co-located is when the sharing is done on one host with several spectators. In co-located spectators can send information to one host visualising the information shared by the spectators. The toolkit should support both types of sharing. The discovery of the other host to share information to is covered by the previous section.

Having the distributed sharing is complicated compared to having a co-located sharing. When distributed sharing is to be done several requirements are raised to the system. The system must have the capacity to carry out the sharing. It also must have effective ways of communicate over network and on different kinds of platforms. The toolkit should implement platform-independent communication protocols for exchanging information. This to make the exchange of information possible between devices running on different platforms, which is normal in a ubiquitous environment.

To communicate with the UbiCollab platform, the toolkit should use the standard interfaces that is defined by the platform. The protocols already in use should be used to communicate over. See section 4.1.2 for more details.

The specific sharing of content should follow well established standards. This can for instance be to use RTP for video-streaming, H.323 for VoIP, etc. I will not go further into this subject, because this is a matter for the developer actually using the toolkit to support such a sharing Service.

How information should be stored in the system depends on the information to be stored. The information to be shared has to be stored in a way fitting to the nature of the content. For instance a messageboard needs to save the messages in a persistent way so that messages not are lost if a the message server goes down. Such a persistent memory can be a database at the Content Server. The choice on how to store the information to be shared is left to the developer because this entirely depends on the nature of the content being shared.

Other information to be saved by Content Server or the Visualising Display is configuration information. This information should be saved in persistent memory in either a configuration file or a database. For a Visualising Display a configuration file should be used. The Content Server should use a configuration file if the information to be saved is fairly static and a database should be used if the information is frequently changed.

5.3.1 Asynchronous versus synchronous information sharing

Research question B-8 is treated here. In addition, the research question B-7 that is about how to support both asynchronous and synchronous information sharing is answered partly here, the rest of the analysis is fragmented throughout the chapter because the answer to the research question affects many aspects in the analysis.

Asynchronous information separates from synchronous information sharing. When asynchronous information is to be shared in the system the system must have some Content Server(s) that can receive and transmit the information independent of if any clients running a Visualising Display are present in the system. With synchronous information sharing the Content Server only has to be running when there exists clients with the Visualising Display in the system that are or want to subscribe to the Content Server's Service(s). Thus, the toolkit should design the Content Server in such a way that it can be run independently of any Visualising Displays. It should also be possible to have Content Servers that are started on demand in situations where this is desired.

Because asynchronous information not is consumed by the receiving end at the same time as it is sent, a developer developing an asynchronous Service should save the information in persistent memory like a database at the Content Server. This way asynchronous information can be kept in the system.

Requirements

R-8: For communication between Content Servers, Visualising Displays, platform-independent communication protocols should be used. The best solution will be to use the same protocols already in use in the UbiCollab platform.

R-9: The configuration information and other information related to the different entities in the system must be saved in persistent memory, either as configuration files or in a database.

R-10: The Content Server should be run independent of any Visualising Displays. It should also be possible to start a Content Server on demand. The Content Server should keep asynchronous shared content.

5.4 Visualising content

In this section research questions A-4, B-5, B-6, C-1, C-2 and C-3 are answered.

The toolkit shall be used to develop ubiquitous shared display applications to be used in different settings and with different purposes. The shared display to be developed can be an asynchronous messageboard, a real-time video streamed conference, a presentation synchronised with several types of devices. This sets high demands on how

the content being shared is to be visualised and to the association between Content Servers and Visualising Displays.

The visualisation of content has to be flexible. A device should not be limited to only show one specific type of content, but have the possibility to show all content that a device is capable of showing only limited by hardware resources. Hence, a device capable of showing some information should be able to access the shared information and visualise it without having to install software on the client previous to the session. The advantages of this are several. First of all a user can access information that is shared with him without worrying if he can access it with the software he has installed. A user can take use of public displays running a client with a Visualising Display and get the content that is shared with him. Public displays can be used by a diverse user group with different needs. All this is only restricted by hardware capabilities and that there exists in the system a Content Server that shares the information wanted.

How to accomplish this visualising of content anytime-anywhere? The Visualising Display is to have a content-pane to show the content that is being shared and the control panels that is needed to control the shared content. Two solutions on how to access content anytime-anywhere are present.

1. The Visualising Display can control the content-pane locally. With this solution the visualising display has to download plug-ins when needed, to visualise the content. For each Content Server the Visualising Display is subscribing to the Visualising Display will download a plug-in that can visualise the content at the Content Server. The plug-ins will be visualised on the content-pane in a way similar to the window concept.

2. The other solution is to let the Content Server control graphic containers that forms a Service. A Visualising Display will load these Services formed as graphic containers remotely. A graphic container can be different types of panels or canvases used to visualise graphics on. This way the graphic containers has to be controlled using some kind of remote controlling mechanism. All interaction with the graphic container will be handled remotely by the Content Server.

Choosing between the two solutions the second one seems like the easiest solution to implement. This solution can easily be implemented using a middleware language. Little configuration and un-complex initial work is to be performed for Visualising Displays and Content Servers with this solution. With the first solution more an advanced implementation is needed. The handling of downloading and running plug-ins dynamically can be complex. Also how to send all semantic information about the content that are being shared can be a complex task. The second solution is therefore chosen as the best solution in this thesis. This because the task of implementing the first solution can become too time consuming. For future work it is left to evaluate if it will be more effective to implement the first solution and if so improve the toolkit by implementing this solution.

In this version of the toolkit only subscribing to one Content Server will be possible. For future versions it should be possible to subscribe to Services from different Content Servers. This way you can tailor your Visualising Display by pick Services. For instance if you want to have a presentation you can pick a presentation Service from the presentation Content Server and a sound conference Service from a sound Content Server. Because the this version of the toolkit will focus on making a toolkit that can provide basic mechanisms for developing a shared display system and the time is restricted this feature will not be implemented, but is left as future work.

Requirements

R-11: The visualisation is to be done by remotely controlling the graphic containers in the Visualising Display. It is left to the Content Server to decide how to visualise the content in graphics containers that shall be on the Visualising Display.

R-12: A Visualising Display can only subscribe to Services from one Content Server at the time.

5.5 Protection of information

The analysis on protection of information is based on research questions B-10 and B-11.

Sharing information with others sets some requirements to protection of the content being shared. Restrictions to information access can be: who can see the content being shared, who can manipulate the information, who can control the information and who can grant the privileges to change protection attributes.

Regarding access to viewing content being shared, it would be natural to grant this right to all members of a collaboration instance. There should be implemented an authorisation procedure that checks if a person or a device trying to subscribe to a Content Server in a collaboration instance is member of the same collaboration instance.

When sharing information it is not always sensible to let the persons with access to view the information, have access to change or control the information sharing. For instance if you are having a presentation. You might want to let the collaboration instance view the presentation, but you still want to be the only one that can control and decide want to be shown in the presentation. It should be a list of members that have the access to control or change the information being shared.

Granting privileges to change the protection attributes must also be possible. The user with default privileges to do this, should be the user adding a Content Server to the collaboration instance or the content that is being shared in the collaboration instance. Some times this person would like to give this privilege to other persons. The toolkit

should implement that the Content Server is maintaining a list of what persons that have the privileges of changing protection attributes.

Another situation that also has to be taken into consideration is the protection of Visualising Displays. When a Visualising Display is used at public screens there has to be some mechanism that makes sure that an active session can not be interrupted. It also must be possible to interrupt default sessions. Thus, there should be possible to set Visualising Displays as interruptible or not interruptible.

Requirements

R-13: The toolkit should implement protection of the content being shared in the Content Server. The protection should be at the service level. Protection should cover read access to content, control and manipulation and granting privileges to change protection attributes.

R-14: There should be possible to set Visualising Displays as interruptible or not interruptible.

5.6 Peer-to-peer versus centralised approach

This section exclusively answer research question B-13.

The arguments for having a peer-to-peer or a centralised architecture are many. The toolkit is to support development of shared displays systems, benefiting sometimes on having a peer-to-peer architecture and sometimes a centralised architecture.

Shared displays systems that could be supported with a peer-to-peer architecture are systems that are shared real-time . By this, it is meant systems that are used to share information by one peer to other peers for instance in a collaboration instance. If a person wants to share a presentation, that person can start up the sharing and other peers can connect.

Other types of applications that need a more permanent and asynchronous Service running, might profit from being run in a centralised architecture. This can for instance be messageboards and blogs. These Services need to run independent of any person using them.

When deciding on having a peer-to-peer or centralised architecture the performance of each alternative must also be considered. The toolkit is to be used for development of shared display systems on top of the UbiCollab platform. The performance has to be scaled to the number of users of the platform.

With a centralised architecture the Content Server can be a bottleneck. This if the the Content Server's Services are popular and/or at the same time are load intensive.

Such systems would benefit of being run peer-to-peer. This way the load will be more distributed in the network. With a centralised architecture overloaded Content Servers can be duplicate to handle high service demands.

Having a centralised architecture makes the Content Servers more complex. They have to be able to serve several sessions at the same time having logic for handling multiple collaboration instances sharing content on one Content Server. In a peer-to-peer solution this would not be necessary. With that solution a Content Server would be dedicated and owned by a collaboration instance.

In many systems the Content Servers must run independent of any Visualising Displays subscribing to their Services. This makes a centralised architecture favourable. The peer-to-peer alternative has some performance advantages that would be good to have. A good alternative to a pure centralised alternative could therefore be to have a hybrid between centralised and peer-to-peer architecture by including application libraries mentioned in section 5.1. This way applications with asynchronous Services could run on centralised servers while applications with synchronous Services could be fetched from centralised application libraries on demand and run in a peer-to-peer fashion. However, this alternative is complex and will be too ambitious for this thesis. A centralised approach is therefore chosen for the toolkit. The hybrid architecture can be developed in parallel with the application libraries discussed in section 5.1.

Requirements

R-15: The architecture of the toolkit should be centralised. This because many of the applications in the system will be asynchronous and therefore needs servers running independent of clients visualising the content being shared in the application.

5.7 Contextualising the information

In this section research questions D-1, D-2 and D-3 will be analysed and given an answer.

In UbiCollab all information is gathered around the concept collaboration instance. A shared display system should also use the collaboration instance as the unifying term. All information that is shared in a collaboration instance should be easy to associate with a shared display if any with capabilities to visualise the information exist in the system.

It is important that DISCOlab uses the existing services in the system. For instance should the privacy service be used to filter content that is being shared and be the service that provides the protection of the content.

An API for communicate with the Content Server must be defined. This API should

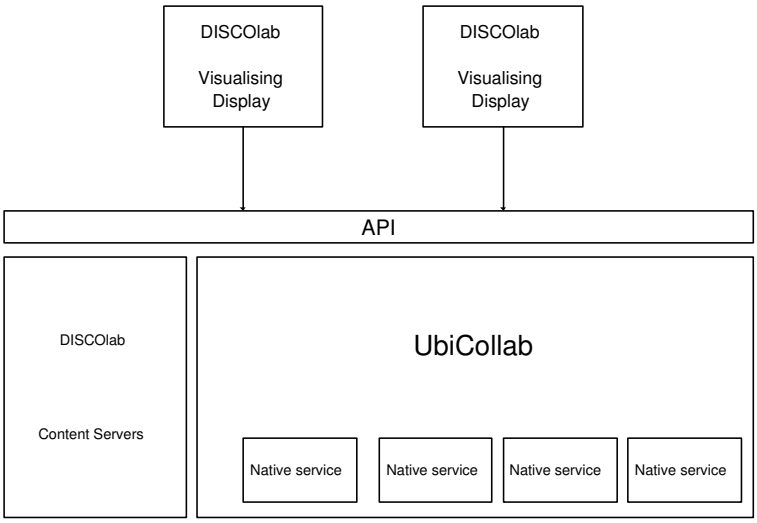

Figure 5.3: DISCOlab: a part of the UbiCollab API

integrate the services that are on the UbiCollab platform. A developer should use this API when he is developing shared display systems on top of UbiCollab. The API should extend the API already present at the platform and keep the UbiCollab API gathered at one place. The DISCOlab should be visible to the users as one of the core services that UbiCollab provide.

With this solution we offer a consistent API located at one place to the developer for all functionality on the UbiCollab platform. A developer can access the DISCOlab functionality and at the same time access the basic services at the UbiCollab platform. It is less complex to contextualise the shared information and get context information from the shared information. We will then have a system as in figure 5.3.

Requirements

R-16: The DISCOlab API should be integrated with the UbiCollab API.

R-17: The basic services of UbiCollab should be used to contextualise the shared information.

R-18: It should be easy for a developer to obtain context information, ie. collaboration instance, presence of users, locations, resources, etc., concerning the shared display application and the collaboration that is taking place in the shared display application.

Summary

In this chapter an analysis has been done on how a toolkit should provide the necessary features to a developer wanting to develop a shared display system for the UbiCollab platform needs. The analysis has been based on the research questions raised in the Problem Elaboration, chapter 3. All research questions have been treated in this chapter. The decisions made in the analysis will form the basis for the design DISCOlab presented in the following chapter.

Chapter 6

Conceptual design

In this chapter the conceptual design of DISCOlab will be presented. The design decisions done should reflect the requirements that were elaborated in the analysis.

6.1 Conceptual model of DISCOlab

From the entities defined in the analysis a conceptual model has been elaborated. Figure 6.1 presents the system entities and the basic communication links. DISCOlab defines three basic entities as described in requirements R1-R3. Also solutions that implements requirements R5, R6, R7, R9, R10, R11, R12, R14 and R15 will be presented in this section.

6.1.1 Content Server

The Content Server is the entity that handles the sharing of some content. The Content Server does this by defining one or more Services that handle and actually do the sharing of some content. When a user wants to use one or more of the Services that the Content Server has, the interested part can download a description list of the services with information regarding the services, address information and what device they are designed for. A description of the Content Server is accessible for clients and what type of content the Content Server support is defined.

The Content Server is independent of Visualising Displays. It will share content in a centralised fashion. The Content Server can be ran from anywhere in the network, as long as it is visible to the UbiCollab platform.

6.1.2 Service

A Service is shared by Content Server. A Service should share the content, control the content and interact with the content. An event log from the interaction with the Service could also be provided. It is in the Service a developer defines the application the developer is to develop. The Service should work as an interface from DISCOlab and UbiCollab to the application. The Service interface will be downloaded by Visualising Displays that will run the application defined in the Service remotely. The application defined in the Service should have a graphical representation that the Visualising Display can visualise.

A Service should in addition to be represented graphically as an application have a UPnP representation that makes it accessible for devices not running Java or not implementing a Visualising Display.

All information that is saved in a Service should be stored in either files or in database. This is left to the developer of the application to decide, because this is highly application dependent.

6.1.3 Visualising Display

A Visualising Display is a client side interface for visualising information shared by the Service. The Visualising Display is a GUI component that handles all interaction and visualising of a Content Server's Services. The Visualising Display can only be connected to one Content Server at the same time. This can in future versions be expanded to support several connections.

The communication between a Visualising Display, and the Content Server and it's Services is abstracted for the developer. A developer should only need to focus on implementing a client that has a Visualising Display, implement an application with a Service interface that is added to a Content Server. The Visualising Display's size and type of device should be sat. Also should the Visualising Display have attributes telling whether the Visualising Display is a public or private display and if it is interruptible or not.

When a Visualising Display downloads a description of the Content Server's Services it will by default visualise the Service it has capabilities to visualise. Restrictions on the capabilities will typically be size of screen, bandwidth and other hardware limitations. The Visualising display should be connected to a Content Server on a session-to-session basis. It will not in this version be possible to subscribe a Visualising Display to a Content Server over several sessions. If this feature is wanted by a developer, this can be developed in the application implementing a Visualising Display.

Configuration information for the entities in the system will be hard coded and/or saved in configuration files.

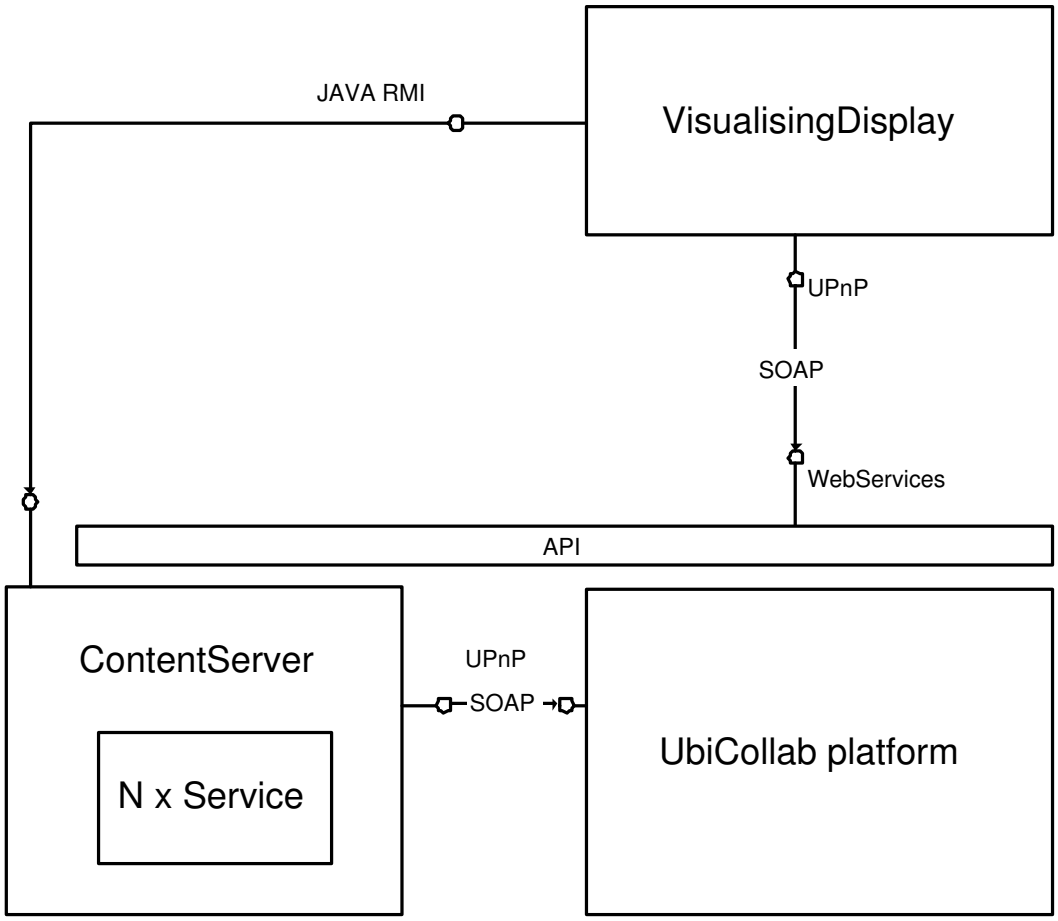

Figure 6.1: DISCOlab overview

6.2 API and communication

In this sections requirements R4, R8, R13, R16, R17 and R18 will be covered.

The entities in the system needs to have ways to communicate and exchange information. As seen in figure 6.1, the Content Server is accessible through the UbiCollab platform API. The same is for the Visualising display. The platform API should defining methods that is relevant for initiation and setup of the subscription of content from a Content Server to a Visualising Display.

Using the web service of the platform to contact Content Servers and Visualising Displays makes it easy to integrate the DISCOlab entities with the rest of the platform. This also makes it easy to use the basic services of UbiCollab to contextualise the information and the entities in DISCOlab. A developer will at the same time only have to deal with one API when using the platform and DISCOlab. When a session is initiated between a Content Server and a Visualising Display the communication will proceed peer-to-peer between the two peers, or a Content Server and a device that gets

the Services over some platform independent middleware protocol.

A Content Server and Visualising Display should have a dynamic discovery mechanism like the UbiCollab services, so that the UbiCollab platform can discover Visualising Displays and Content Servers dynamically.

DISCOlab should use the basic services to contextualise information and the entities. For instance, a Content Server must be member of a collaboration instance to be subscribable to a Visualising Display, the same is for Visualising Displays and the information that should be shared by a Content Server. The location service gives location information on Visualising Displays. For protection of information, DISCOlab should use the privacy service to filter the information and entities of DISCOlab. The requirement that the entities must be members of a collaboration instance, makes it easy to access the context information that a sharing session should have.

6.2.1 API

The DISCOlab API should be located with the UbiCollab API at the same web service. This will make it easier to contextualise the information and also will it be better for developers using UbiCollab. The API should provide the following basic DISCOlab functionality:

- Searching for a Content Server that supports sharing of some specified content.
- Adding and removing content to a Content Server.
- Subscribe and unsubscribe a Visualising Display to and from a Content Server.

For the rest of information retrieval and protection the basic services of UbiCollab should be used. The privacy service will filter the information and content that is to be shared by a Content Server. The location service will handle search queries such as “find closest Visualising Display to user X” and give location information through user and resource profiles. For getting information in one context, the collaboration instance will provide this information. The collaboration instance will also provide information on the users and resources associated with it. Furthermore it will give presence information on the users. Putting the DISCOlab entities in a contextualised setting is done in the DISCOlab API. To access the API, information relevant to the user-session and collaboration instance must be submitted. This information will be used to deduce contextual information relevant to a sharing session.

Summary

A design of DISCOlab is presented. This design is to be implemented so that a developer can use DISCOlab to easily develop shared display applications on UbiCollab.

Chapter 7

Prototype

In this chapter I will present the prototyped version of DISCOlab with the implemented solutions. I will present important interfaces and key objects in the architecture to explain how they work and how they shall be used to develop a shared display system. I will also state the reason for my solutions. First I will give a short description of the technology used, then I will explain the Content Server and Visualising Display along with the components. Finally, I will describe how the interaction between the Content Server and Visualising Display is done.

The prototype implements the architecture presented in the design. A Content Server has been created with a Service that defines the application. Furthermore a Visualising Display that can display the application defined by the Service is implemented. DISCOlab is implemented as a proof-of-concept and more work has to be done to make it a fully functionable and error handling system.

7.1 Technology

For the implementation of DISCOlab, Java 2 Standard Edition version 1.4.2 [J2S] has been used. Today Java can run on almost any device and is thus more or less platform independent. Java has also been used in the implementation of the UbiCollab platform and it was natural to continue the use of Java in the platform.

To make the Content Server and Visualising Display dynamically discoverable, the Universal Plug and Play (UPnP) [UPn] framework has been used. The CyberLink UPnP API [Cyb] provided the functionality to do this. The motivation behind using UPnP for dynamic discovery is the fact that UPnP has been used in previous UbiCollab project and is the only dynamic discovery mechanism supported by the platform. UPnP uses SOAP [SS00, SOA] for communication. SOAP is platform independent and thus contributes to the arguments for using UPnP. To extend DISCOlab to support other discovery protocols will not require much work and can be done if it is necessary.

Since a Visualising Display should support to display any type of information without pre-session configuration and installation of software, a way of dynamically download program code and run it on the fly was needed. To do this Java Remote Method Invocation (RMI) [RMIa] was used. This because Java already was in use to implement the software and also because Java RMI has the functionality needed and that it is platform independent.

The communication with the UbiCollab platform is done over the web services at the platform.

7.2 Content Server

The Content Server is the server side part of DISCOlab. It is here the shared display application should be defined and where all synchronisation and sharing of information should take place. The Content Server is built up around two important interfaces, the ContentServer and the Service, see figure 7.1. I will first describe the Service and then the Content Server.

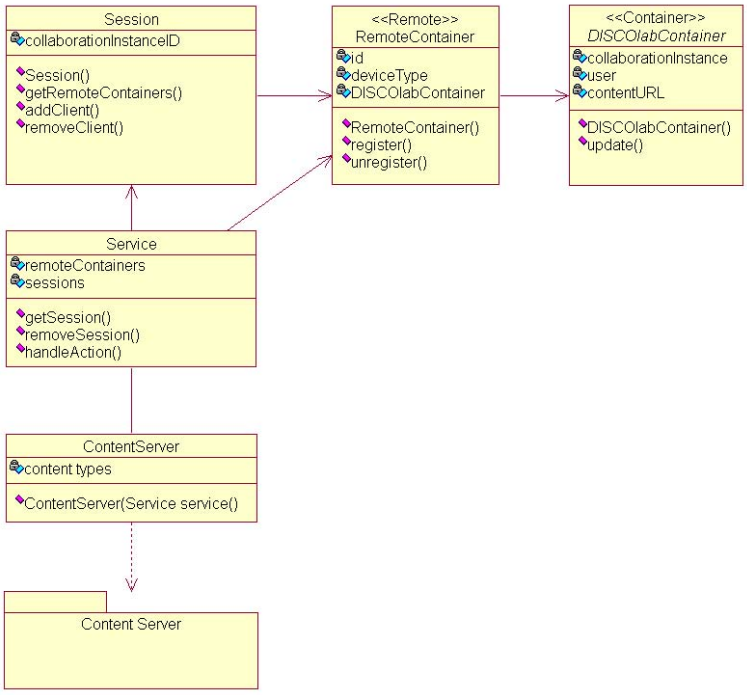

Figure 7.1: The Content Server interfaces

7.2.1 Service class

The Service class is the entity and interface where the developer shall define the shared display application. The Service class is an abstract class that will be a super class for the application core class. From the Service class the developer can access and set all necessary information to handle sessions with the different collaboration instances that use the application, get information for each user, set up configurations that a Visualising Display can access to personalise the application, add one or more visual representations of the application, etc.

The Service has several important classes that it uses and is dependent on to distribute the application developed. These are RemoteContainer, DISCOlabContainer and Session, see figure 7.1. The RemoteContainer is used by Java RMI as communication medium. The RemoteContainer includes a DISCOlabContainer and is downloadable by clients. The DISCOlabContainer is a graphical component that serves as the interface for the shared display application. It is an abstract class that will be implemented by the graphical front-end of the shared display application. The DISCOlabContainer can be downloaded through the RemoteContainer and be visualised by the VisualisingDisplay. A shared display application can and should support several device types with different display capabilities. For each display that is supported by the application, a developer should implement a new DISCOlabContainer and define it in a separate RemoteContainer. In the RemoteContainer the developer can specify the device type. When a Visualising Display is subscribing to a Content Server it will download the corresponding RemoteContainer that has a DISCOlabContainer with the graphical interface suitable for the device type.

The Session class is the class that handles the active Sessions in the Service and ContentServer. The class is used by the Service for routing of messages and updates to the correct collaboration instance.

7.2.2 ContentServer class

The ContentServer class is the part of the Content Server that handles all application independent functionality in the Content Server, in contrast to the Service class which is highly application dependent. The ContentServer class is created with the mission to support the Service class in distributing the application and handle communication. It is the interface for the ContentServer package that can be seen in figure 7.2. The ContentServer class takes care of creation of new sessions that are created when a collaboration instance needs the services of the Content Server. It also prepare the RemoteContainers in the Service class to be downloaded by clients.

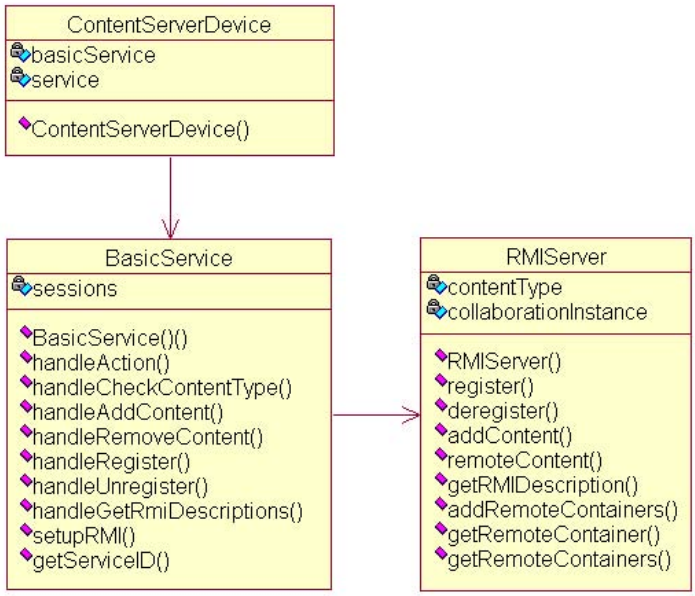

Figure 7.2: The Content Server package

In figure 7.2 the UPnP interface, ContentServerDevice, can be seen. This class is the central class for the package. It sets up UPnP communication and defines the BasicService. The BasicService is the class that handles communication with the UbiCollab platform and that is implementing the functionality in the Content Server API. Furthermore the package is defining the class RMIServer. This class is responsible for making the shared display application defined in the Service downloadable for clients.

The Content Server API is accessible through the UbiCollab API and is used by the UbiCollab API to initiate and set up sessions with the Content Server. The Content Server has the following interfaces to support this:

- checkContentType: Tells if the Content Server supports the submitted content type or not.
- register: Registers a collaboration instance to the Content Server.
- unregister: Unregisters a collaboration instance from the Content Server.
- addContent: Adds the content to a collaboration instance session in the Content Server, specified with a url.
- removeContent: Removes the content from a collaboration instance session specified in an url.
- getRMIDescription: Returns a xml description of the Content Server's applications and information needed to connect and download the applications.

7.3 Visualising Display

The Visualising Display is the client side part of DISCOlab. A Visualising Display can be implemented by a client and displayed. It has a graphical front-end that is visualising the application that has been subscribed to and downloaded from a Content Server. The subscription to a Content Server can be invoked remotely or locally through the UbiCollab web service.

In the figure 7.3 the components of a Visualising Display can be viewed. The class VisualisingDisplay is the interface class that visualises the shared display application that is downloaded to the client. The class is a graphical container. The VisualisingDisplay class lets you configure it with user settings and information about the collaboration instance, both previous to a session but also during the session. The VisualisingDisplay feeds the downloaded DISCOlabContainer with this information so that the downloaded shared display application can get access to the user information and contextualise and personalise the application to the user. It also has methods to update the Visualising Display. A Visualising Display can be sat to be updated with a preferred refresh rate.

To support the functionality in the Visualising Display several components has been created as seen in figure 7.3. The VisualisingDisplayDevice is a UPnP enabled device that is the central component in the Visualising Display. The VisualisingDisplayDevice advertises itself to the network and starts up a UPnP service, the BasicService, that will handle the communication with the UbiCollab platform. It also creates a RMIClient that can download Java RMI objects, ie. RemoteContainer, and export a VisualisingDisplay to the RMIServer specified in the Content Server.

The BasicService specifies a few methods that functions as the interface to the Visualising Display and that supports remote invocation of sessions with a Content Service. These are as follows:

- subscribeToContentServer: Receives a url and description of a Content Server and connects the Visualising Display to it. Initiates download of the application distributed by the Content Server.
- unsubscribeToContentServer: Teares down the connection with a Content Server and resets the Visualising Display so it is prepared for a new session.

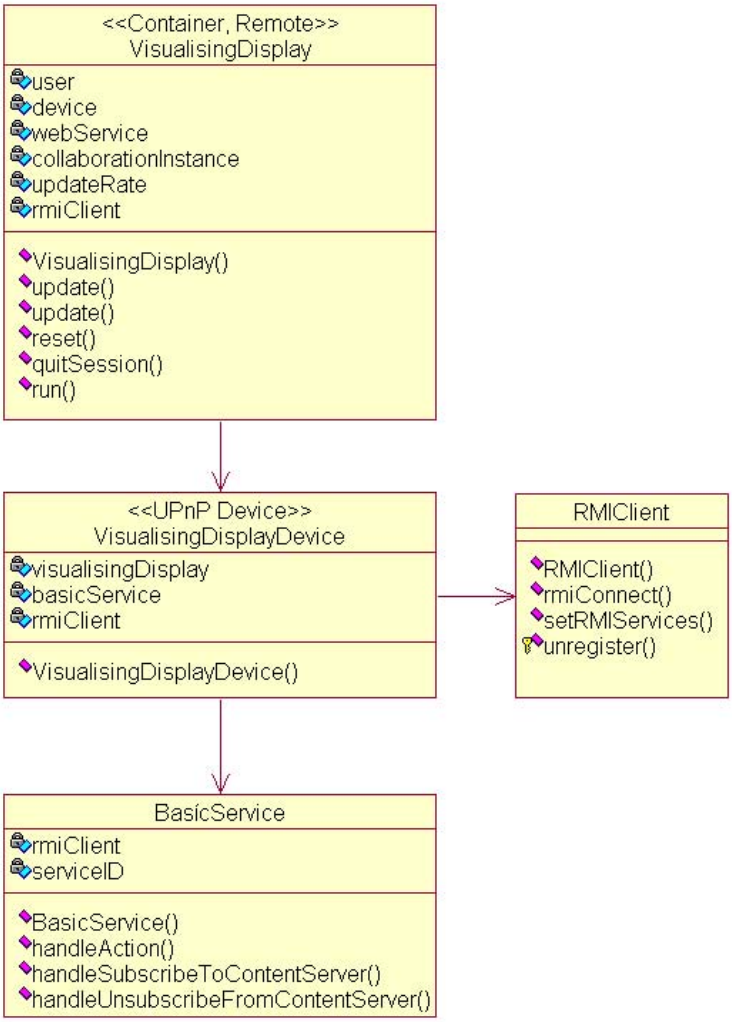

Figure 7.3: The Visualising Display classes

7.4 Interaction between the Content Server and the Visualising Display

To enable communication between Content Server and Visualising Display, remote invocation of the VisualisingDisplay, and to download the application from a remote location, several steps have to be taken in the system. I will in this section introduce to the reader how this is done in DISCOlab.

To initiate a session between a VisualisingDisplay and a Content Server the steps illustrated in figure 7.4 are followed. A client gets access to the entities through the UbiCollab web service. See section 7.5 for the web service methods for doing this. First a user searches for a Content Server that supports the content to be shared. If found, the user adds the Content Server to the collaboration instance, before the user

connects a client running a Visualising Display to the Content Server. The user can use the UbiCollab web service to search for a client running a Visualising Display. The search can have different criterias such as location or name of Visualising Display. After the Visualising Display has subscribed to the Content Server, the application version at the Content Server that the Visualising Display can display will be sent to the display and visualised at the screen.

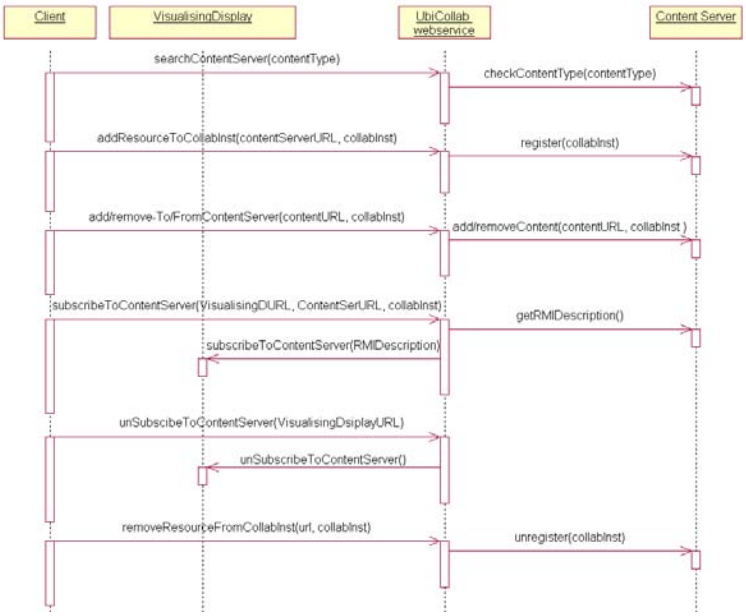

Figure 7.4: Initiation of a session using UbiCollab web service

When a Content Server gets a request to register a new collaboration instance, it registers RemoteContainers at the RMIServer for that collaboration instance, see figure 7.5. When a Visualising Display is asked to subscribe to a Content Server in BasicService, it forwards the RMI description xml file to the RMIClient which in turn connects to the RMIServer. When the RMIClient connects to the RMIServer it downloads a RemoteContainer, using Java RMI. The RemoteContainer lets the Visualising Display register itself to the RMIServer that adds the Visualising Display to the Session object of this collaboration instance (each collaboration instance keeps a session object). The Session object updates the Visualising Display by adding the shared display application graphical front-end, the DISCOlabContainer, to the Visualising Display. The DISCOlabContainer is sent to the Visualising Display and visualised at the client's screen.

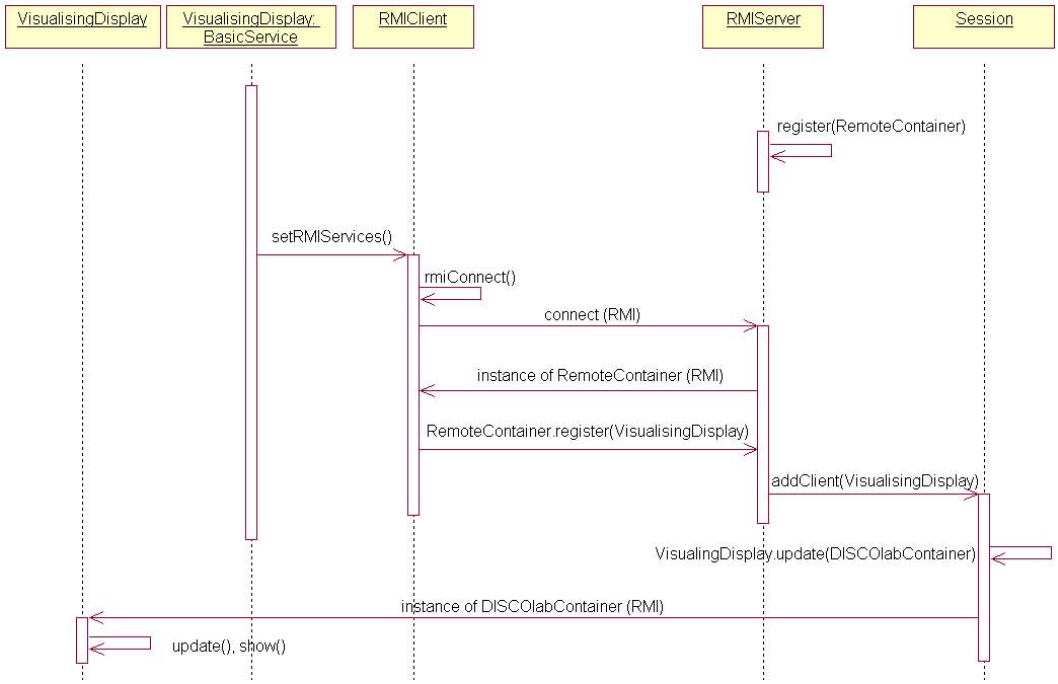

Figure 7.5: Communication between Content Server and Visualising Display

7.5 Integration with UbiCollab

As mentioned above, the UbiCollab web service is used to initiate all sessions between Content Servers and Visualising Displays. The UbiCollab web service will provide access to DISCOlab entities at the same level as other services and components of UbiCollab. This makes it easy for developers to use information and get context information related to DISCOlab.

This is the UbiCollab web service that should for communication with DISCOlab entities:

- searchContentServer: Searches for a Content Server that can share content of a specified type or functionality, eg. PDF files or whiteboard.
- addContentToContentServer: Adds content to a Content Server, eg. a PDF file that should be shared.
- removeContentFromContentServer: Removes some specified content from the Content Server.
- sunscribeToContentServer: Subscribes a Visualising Display to a Content Server.
- unSubScribeFromContentServer: Unsubscribes a Visualising Display to a Content Server.

The developer should use the web service of UbiCollab to get contextual information about the entities built on DISCOlab. Such information can be; what collaboration instances does Content Server X be a member of? Where is the closest Visualising Display to user Y? An overview of the Web services can be found in appendix C.

DISCOlab offers a class, WebService, for communicating with the Web service at UbiCollab. In figure 7.6, the WebService class along with the classes CollaborationInstance, Person and Resource can be seen. The last three classes instantiate: a collaboration instance, a user or person in UbiCollab, and a resource, which can be a physical or logical resource, eg. a PDA, PDF document, etc. The collaboration instance will keep information on all resources and persons that are members of the collaboration instance.

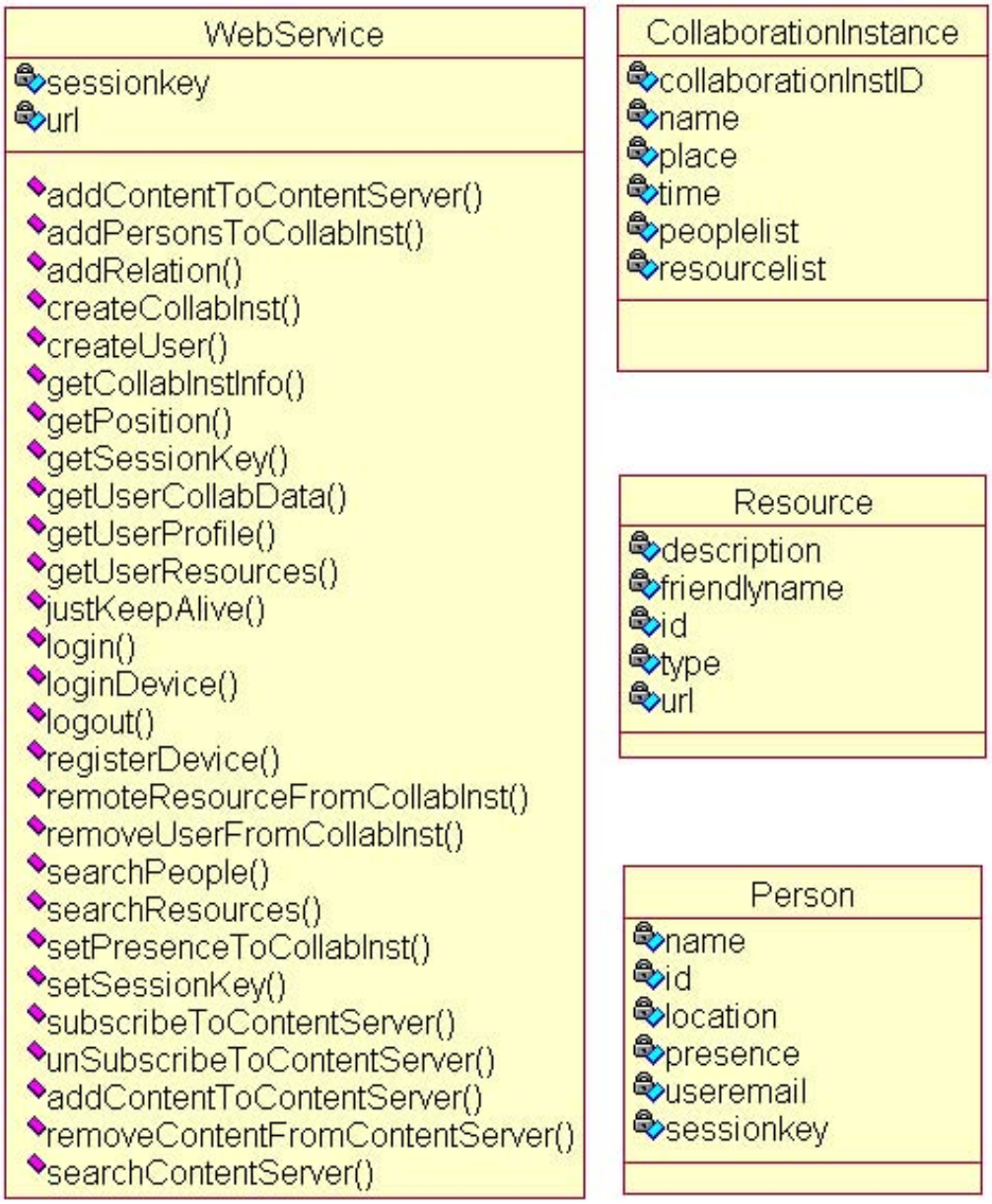

Figure 7.6: Classes for UbiCollab communication and instantiations

Summary

In this chapter I have presented the prototype of DISCOlab. We have seen that a toolkit is developed that can be used to implement shared display applications on the UbiCollab platform. DISCOlab can be accessed through the UbiCollab web service. In the next chapter I will demonstrate how DISCOlab has been used to create a shared display application.

Chapter 8

Demonstration

To demonstrate how DISCOlab can be used to create a shared display application, a Messageboard has been developed. I will in this chapter first describe how I used DISCOlab to implement the Messageboard. The Messageboard was implemented to verify that a shared display system can be implemented using DISCOlab, and to illustrate a relevant subset of features DISCOlab provide to the developer. The Messageboard is only implemented for a personal computer, while DISCOlab is designed to support other devices as well with the only requirement that the devices run Java. I will therefore give a hypothetic description of how DISCOlab would be used to extend the Messageboard, to give support to a PDA, to illustrate that a DISCOlab application can be extended to give support to several types of devices.

To demonstrate that DISCOlab can be used for other types of applications as well, I will present a hypothetical drawingboard application and describe how a developer would use DISCOlab to implement it.

8.1 Messageboard application

The demonstrator is demonstrating one type of shared display application that can be developed on DISCOlab. The dimensions the demonstrator follows are: asynchron communication, that is both co-located and distributed and it supports informal collaboration. By co-located communication I mean that the Messageboard will give support to users that are co-located, but not present at the location at the same time.

First I will describe the Messageboard and how it is implemented on a Content Server, then I will give a description of a client implementing a Visualising Display for visualisation of the Messageboard.

8.1.1 Messageboard on a Content Server

The Messageboard application is an applications that gives users of a collaboration instance the possibility to send messages to the collaboration instance from different locations. In figure 8.1 the Messageboard can be seen in the Visualising Client. The Messageboard is implemented for personal computers.

As the reader can see in figure 8.1, the Messageboard application has three parts. These are from upper left the collaboration instance section of the window, the message list and finally the message input section in the lower right corner.

Under the DISCOlab Messageboard logo, the user that is logged into the client is displayed. The user's presence in the collaboration instance is also displayed to the user, so that the user knows how he is perceived by the other members of the collaboration instance. The presence information on the user is deduced from the collaboration instance, since the collaboration instance is the context in which the user is. Finally the location of the user is presented. The location is to be queried from the location service at the UbiCollab.

The context information that is provided in the Messageboard is accessed over the web service interface at UbiCollab. This is done through the WebService class that is provided by DISCOlab. The results of the queries is used to build objects representing entities in the UbiCollab platform, like Collaboration Instance, Person and Resource. Location information should be access through the Person object. Presence information is deduced from the Collaboration Instance in which a person is member of.

I will point out that this demonstrator has the purpose to demonstrate also how contextual information can be gotten from the UbiCollab platform and used by DISCOlab. At the time present the services presence service and location service is not fully developed, thus the location- and presence information is not fully obtainable. Therefore, for the demonstration, fake locations and presence values are feed into the configuration of the Visualising Display to illustrate how the information could be used if the service was fully functional. All other information is obtained from the UbiCollab platform.

The collaboration instance is showed in the top left corner section. Here, all the members of the collaboration instance is visible. For each person in the collaboration instance the user can see the name, presence and location. Also the colour on each person will change dependent on what presence the user has. For this application there are three presence levels; offline which is indicated by red, away which is indicated by blue and present which is indicated by black.

In the message list, the messages will be displayed along with which user submitted it and what time the message was sent to the Messageboard, see figure 8.2.

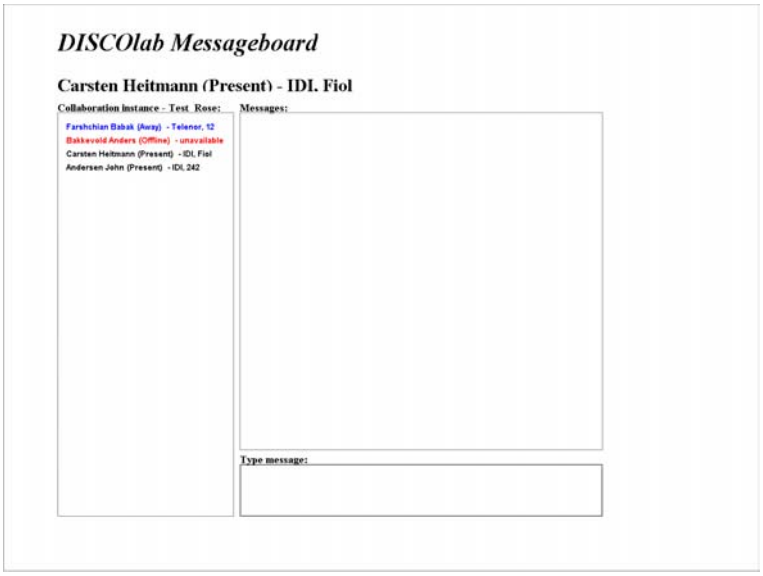

Figure 8.1: The DISCOlab Messageboard

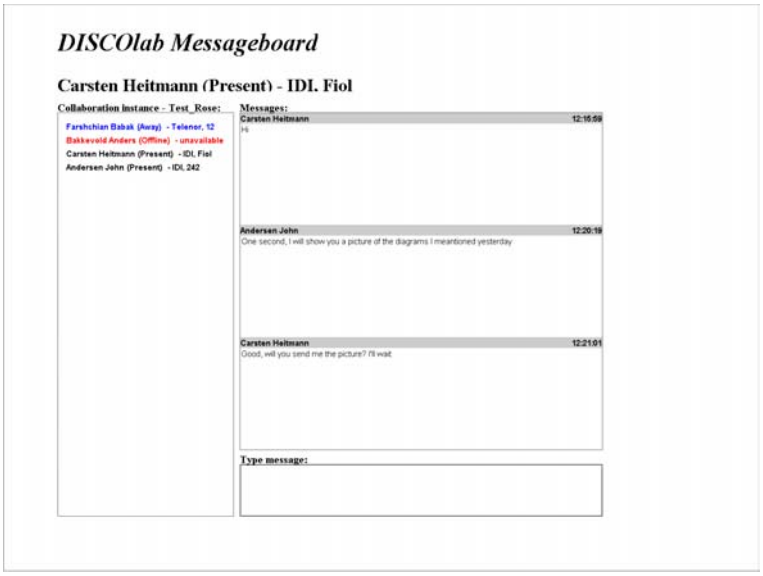

Figure 8.2: The DISCOlab Messageboard after sending messages

Implementing the Messageboard using DISCOlab

When implementing the Messageboard I had to follow the following steps.

As seen in figure 8.3 I first created the classes that realised the Service and the DISCOlabContainer. These was the MessageBoardService and the MessageBoardPanel. From each of this classes I developed the Messageboard. The MessageBoaredService is the class that handles all updates and forwards messages to the correct collaboration

instance while the MessageBoardPanel is the application's graphical front-end.

The Messageboard sends the messages over UPnP using SOAP. Each message is received at the Content Server and handled by the MessageBoardService. The MessageBoardService looks up the correct Session using the collaboration instance as identifier and updates the correct MessageBoardPanel. The MessageBoardService works as a server and the MessageBoardPanel as a client or client side application.

A question that the reader must have raised by now is how the Messageboard gets the collaboration instance information and the user information and how this can be updated in the Visualising Display. The DISCOlabContainer has an abstract method that the MessageBoardPanel realise. This is a update method. In this method all dynamic updates will be done. This method is called by the Visualising Display every time it is refreshed.

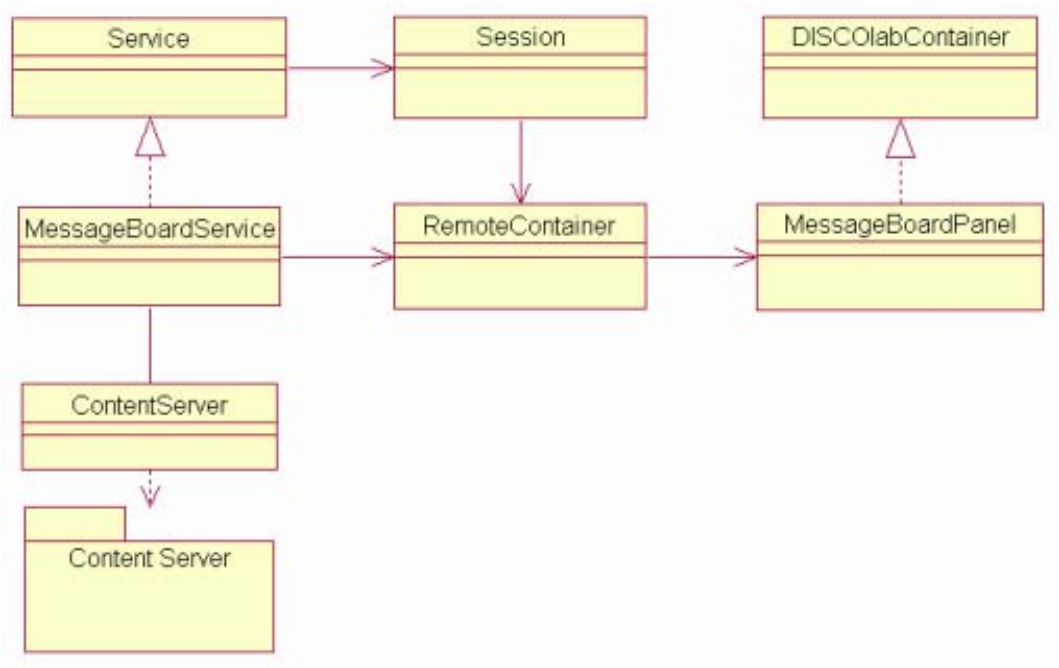

Figure 8.3: The DISCOlab Messageboard architecture

8.1.2 Client with Visualising Display

The client is a graphical window that has added the Visualising Display as it's only component. It will be ran at a personal computer and will because of that only be capable of showing DISCOlabContainers that are accessed through a RemoteContainer that is accessible for devices of type personal computers. The client will after startup, be waiting for remote invocations. The client has not implemented any features for login or local controlling, although this is supported by the Visualising Display. This because I consider it out of the scoop of this project and that the time available is limited.

The login to the client will be done from a remote location, using a UbiCollab client that has the needed capabilities. The information submitted from the remote location like the user that logs into the client and what collaboration instance the user is using the client in, will be used to configure the Visualising Display. This will in second hand be used by the Messageboard to contextualise the application for the user. This is done through previous mentioned interfaces like the DISCOlabContainer and the Service, see chapter 7.

In figure 8.4 the default DISCOlab screen in a Visualising Display can be seen, previous to a session with a Content Server.

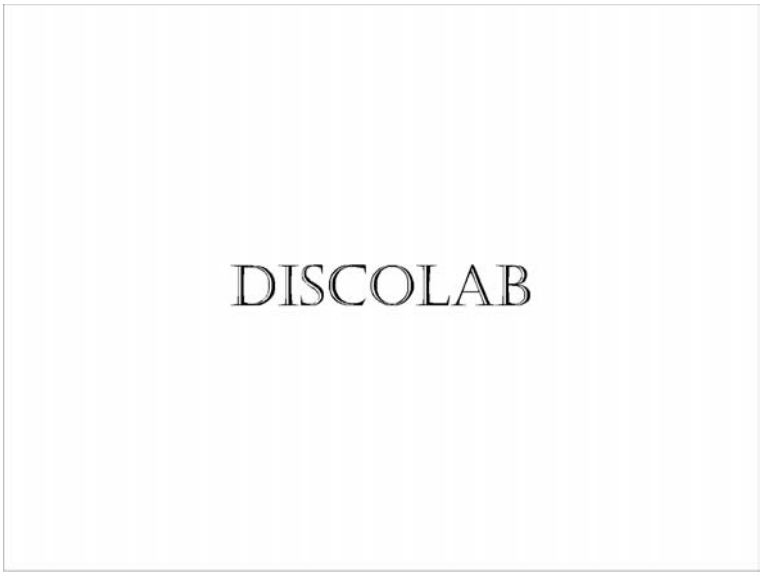

Figure 8.4: The default DISCOlab screen

I emphasise that the client with the Visualising Display is a generic client for all types of DISCOlab applications, ie. the client is not implemented to support only the Messageboard, but will visualise any application that has implemented the interfaces defined at the Content Server, the DISCOlabContainer and the RemoteContainer.

If the client was to switch between or from another application, this is done by subscribing to the Content Server implementing the other application.

8.2 Extending the Messageboard, giving support for a PDA

In this section I will describe how a developer would proceed to implement support for a PDA, in the Messageboard, by using DISCOlab.

Before the PDA can visualise any content from a Content Server, it needs to implement the Visualising Display. This can be done by adding the Visualising Display to a graphical window at the PDA, using Java. The PDA also needs to have the Java 2

Platform, Micro Edition (J2ME) [J2M] and the J2ME RMI Optional Package, (RMI OP) [RMIb] installed to run DISCOlab. Now, let us say that this is done and that we want to add functionality to the Messageboard, so that a Visualising Display at a PDA can subscribe to it.

In figure 8.5 we can see how the Messageboard could look like at the PDA. The Messageboard is looking similar to the version implemented for the personal computer, but is scaled down to PDA screen resolution and can handle other input devices than a personal computer.

The Messageboard is already implemented in the Content Server. What is missing is a GUI application front-end that is adapted to the PDA. The GUI application front end should be created as a sub class of DISCOlabContainer. The GUI should be built using standard Java Swing components and be display in an appropriate way for a PDA. The GUI needs also take into consideration the input devices of the PDA and use listeners that communicate with these. The application logic does not need to be changed, since this is independent of the device type. When this is done, the class implementing the DISCOlabContainer should be added to a RemoteContainer with device type sat to PDA. After the RemoteContainer and the DISCOlabContainer classes are implemented, the RemoteContainer can be added to the Service that is added to the RMIServer by the ContentServer class.

When this is done, the PDA should be capable to subscribe to the Content Server having the Messageboard, and use the application in a similar way as the personal computer.

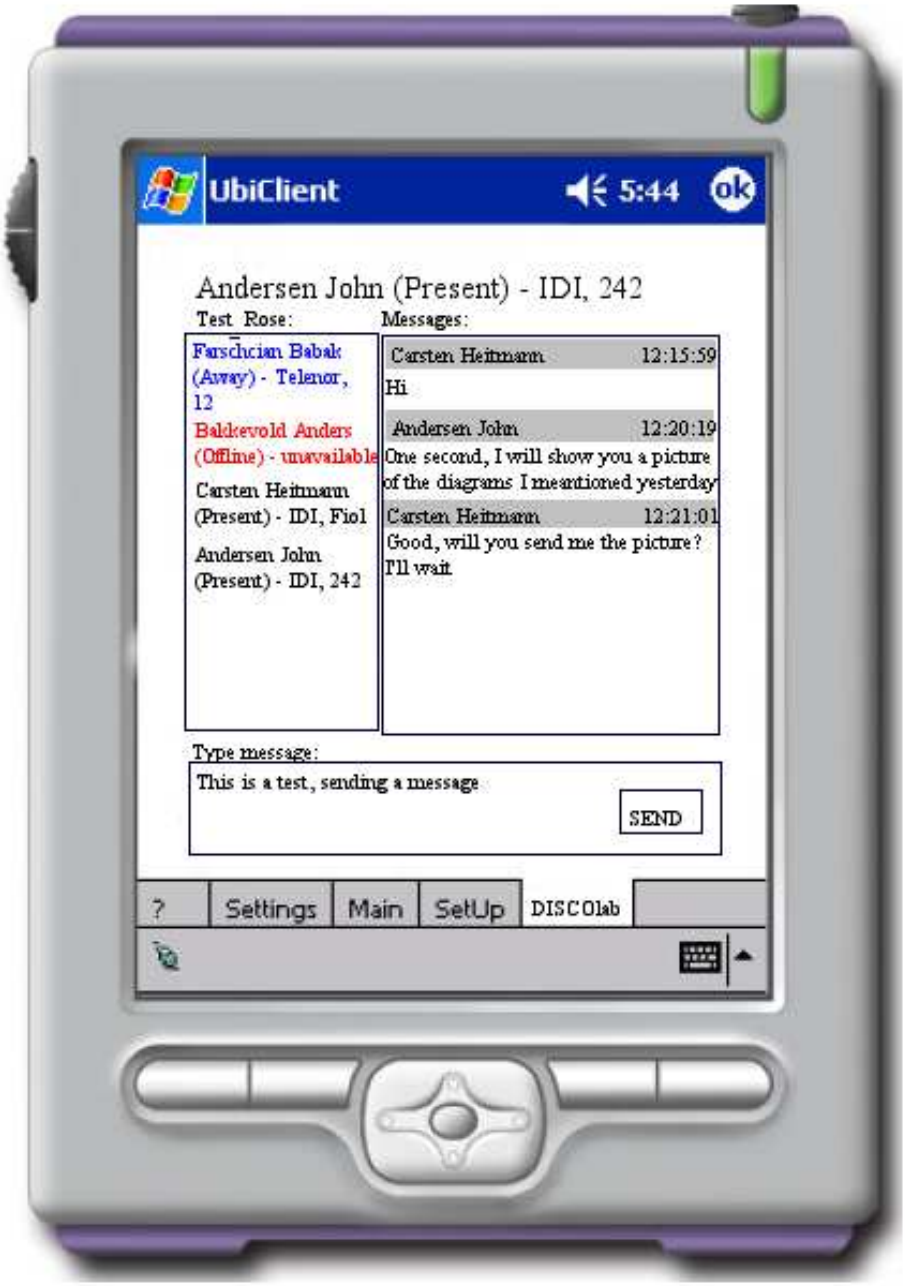

Figure 8.5: Messageboard for a PDA

8.3 A drawingboard application

The Messageboard is an application that is asynchronous and mainly giving support for informal collaboration. It can be a trigger for more formal collaboration by providing a medium that people can switch to more formal collaboration by using, for instance a

drawing tool to illustrate some ideas.

I will in this section describe how DISCOlab would be used to implement a drawingboard. The reason for which I choose a drawingboard, is that this in contrary to the Messageboard is a synchronous application, giving formal support. The drawingboard can be used both by co-located and distributed parties, while the Messageboard would mainly be used by distributed people or co-located, but asynchronous - at different points in time - present at the location of the Messageboard. By using the drawingboard and the Messageboard as demonstration I will illustrate that DISCOlab can be used to develop applications following all the dimensions discussed in chapter 2.

In figure 8.6 we can see how the drawingboard can look like on the DISCOlab client. To implement the drawingboard on a DISCOlab Content Server the following steps must be taken. For the drawingboard itself a free-ware application could be used. The drawingboard should be added or created as a DISCOlabContainer and the necessary steps must be taken to integrate it with DISCOlab, like adding it to a RemoteContainer that is handled by a Service implementation.

In the Service subclass implementation, let us call it DrawingBoardService, the synchronisation of the different drawingboards has to be done. The Service manages a list with all the sessions on each collaboration instance. From this list the Service can get all the clients subscribing to the drawingboard server, and hence update them when events occur in the drawingboard. The DrawingBoardService should have a thread checking all the drawingboard clients if there have been any updates. An update is set as a flag in the drawingboard at the client side. This can be checked by the DrawingBoardService. When an update is detected, the DrawingBoardService will update all other Visualising Displays in the collaboration instance that is subscribing to the drawingboard.

Since the implementation of the drawingboard has not been done, it is difficult to say that this is a working example. However, say that the implementation is done and working, we will have two DISCOlab applications running on two Content Servers. A collaboration instance that would like to use the applications would be able to search for Content Servers that can send messages and that the users can make illustrations on.

Say that there are two users having some informal collaboration on the Messageboard and they decide to follow up on some ideas that one of them has. He wants to illustrate his idea and suggest that they use the drawingboard. He will search for the drawingboard, and send it to the Visualising Display of his colleague and his own Visualising Display. When he has made his illustrations they can switch back to the Messageboard and go on discussing his idea.

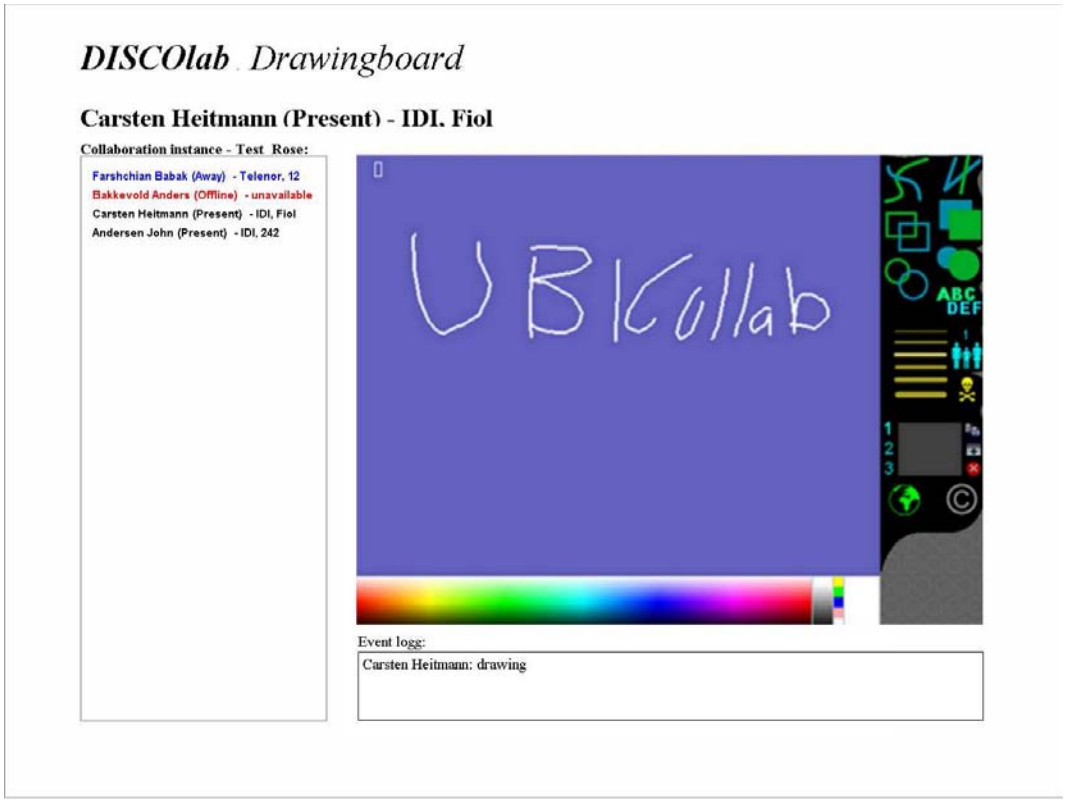

Figure 8.6: A drawingboard application

Summary

In this chapter a demonstration of what DISCOlab can do, has been done. The demonstrator shows that a Messageboard application can be implemented and run as a Content Server. The Content Server is dynamically discoverable by the UbiCollab platform. A client to the platform can access the Content Server through the UbiCollab web service.

With the Messageboard demonstrator I have verified that DISCOlab is a working implementation of the toolkit. I have illustrated how DISCOlab can be used to add support to the Messageboard to support not only visualisation of the Messageboard on a personal computer, but also on a PDA. This shows that DISCOlab can support several types of devices.

Furthermore I have discussed how DISCOlab can be used to create a synchronised drawingboard. This has been done to show that DISCOlab support not only creation of one application, but can support implementation of several types of applications, covering the dimensions of a shared display discussed in chapter 2.

The Messageboard shows that DISCOlab contextualises the information. The context information is not lost when switching from one application to another, ie. from the

Messageboard to the drawingboard.

Finally, we can see in the demonstrator that DISCOlab can dynamically be detected by the UbiCollab platform. Users can search for Content Servers that have the needed functionality when the user needs it. A device running a client with a Visualising Client can be detected by the UbiCollab platform, and a user can search for a Visualising Display by location and get the information on the closest Visualising Display. The user can display all information that is supported by a Content Server in the UbiCollab environment on that display, as long as it is supported by a Content Server.

Chapter 9

Evaluation

The thesis resulted in the development of a toolkit called DISCOlab, that supports the creation of shared display systems on top of the UbiCollab platform. A prototype has been developed, see chapter 7, based on a conceptual model, see chapter 6. A shared display system was developed as demonstrator, see chapter 8, using the prototype to demonstrate and evaluate how the toolkit could be used to create a shared display system on UbiCollab.

In the introduction chapter the following expected results for this master thesis were defined:

- A toolkit for developing shared display systems on UbiCollab, with focus on providing access to the UbiCollab services, should be developed.
- A demonstrator of a shared display system will be developed, using the toolkit. This demonstrator will evaluate how the toolkit can be used to develop a shared display system running on the UbiCollab.

In DISCOlab the focus has been to create a flexible infrastructure to build shared display systems on. This has been achieved by building the toolkit on the UbiCollab. By using UbiCollab we have seen that contextualisation of information has been achieved. Furthermore, DISCOlab has provided UbiCollab support to: let users of the platform to get access to shared displays where you need them - when you need them, eg. send the drawing board to your colleague's screen so he can illustrate what he means, and developers to build such systems on top of UbiCollab without focus on low level implementation issues, ie. communication, session management, device identifying, etc. DISCOlab does not give specific support to any type of shared display application in itself, ie. there is no tools in DISCOlab that supports you in implementation of typical shared display features like multiple mice pointers, synchronising screens, etc.

Typical shared display features have not been the focus of several reasons. Because DISCOlab was to support so many different types of shared display applications, the

domains would be too large to provide specific support for all types of shared displays. Furthermore, many toolkits exists today that give this support, so this functionality was already provided. The idea is that developers instead can use the toolkits to implement the support in the application and integrate the application on DISCOlab that will provide the developer with a infrastructure supporting shared displays and contextualisation of information, see figure 9.1. If DISCOlab was to give support to specific shared display systems, the scope of support that DISCOlab would provide would be too wide and had to be reduced, for instance only giving support to synchronous real-time applications for formal collaboration.

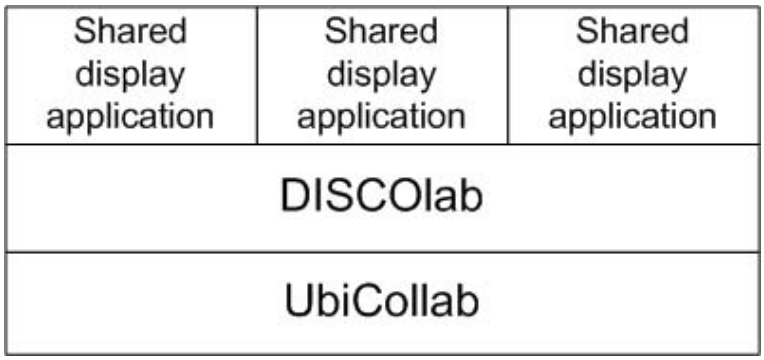

Figure 9.1: Shared displays in DISCOlab

9.1 Research method

The research method in the project has been scenario based and prototype driven. Scenarios have been used to elaborate a set of research questions to build the analysis and the design. A toolkit prototype was developed and a demonstration system was used to evaluate the toolkit prototype.

Using scenarios to drive the research has been a method working well. The scenarios have illustrated many aspects of shared display systems. They have been a driver to identify requirements and functionality that the toolkit needed to support in different types of shared displays. Furthermore, the scenarios put UbiCollab in perspective to the toolkit. However, the scenarios might not be covering important aspects that should have been looked into and thus have been overlooked since the focus has been on the scenarios.

Using a demonstration system as an evaluator of the toolkit prototype has been an effective way of evaluating the prototype. Using the prototype to create a demonstrator highlighted how the prototype was in use and short-comings that it had. It has been verified by implementation that the prototype can be used to implement a shared display system. Most of the work when developing the demonstrator has been lied down in GUI issues, even though some time was used on bug fixing in the prototype and adding some low level functionality, like better support for dynamic updates, better session management, etc. However, the evaluator had limited functionality, ie. not

covering all dimensions of shared displays discussed in chapter 2. Because of this, it has not been proven that the prototype gives support to functionality not covered in the demonstration system.

9.2 The conceptual model

The research questions where all addressed in the analysis. The analysis raised 18 requirements to DISCOlab, see chapter 5. All requirements have been looked into and have been included in the design of the toolkit.

The conceptual model developed has been experienced as a good solution when implementing the prototype. The division into the defined entities, Content Server, Service and Visualising Display, gives support to separate the functionality in DISCOlab and provides basic components that UbiCollab recognises, ie. the Content Server and Visualising Display are incorporated as components in the UbiCollab platform and in the platform's web service.

DISCOlab has a weakness that is does not provide any peer-to-peer connectivity between screens and mobile devices. Visualising clients should have capabilities to communicate with devices when they come into range, over protocols like Bluetooth [Blu], similar to the Speakeasy system presented in section 4.4. This so that the mobile devices could be used to control the screen and initiate sessions at the screen.

9.3 The prototype

The prototype implements the conceptual model presented in the design, chapter 6. However, some issues have been left out in the implementation because of limited time for implementation, making it important to focus on core functionality.

In section 4.3, a list of toolkit properties was presented. DISCOlab implement most of these properties:

- DISCOlab has been implemented in a familiar and well known programming language, Java.
- It removes low level implementation burdens like session management, communication, adaptation to UbiCollab, data sharing, etc.
- It presents itself through a concise API and makes things achievable in a few lines of code, things that would otherwise be complex.

I will say that DISCOlab satisfies the properties well. However, the programming API could be more intuitive and easy to use. Because of the experience level, some unforeseen implementation issues came up. This this can have lead to that the functionality

has been spread on too many interfaces in the toolkit. It can also be confusing for a developer to know what information that is dynamically sat in the Visualising Display and how to implement this and use it in the shared display application that is in development.

Creating a Visualising Display that would run at a client has been a good solution. This way the developer do not need to alter clients when a new shared display system is developed. A client can be implemented with as many Visualising Displays as needed. This way a client can subscribe to several Content Servers at the same time, eg. if a user wants to use the Messageboard and the drawing board at the same time.

However, some functionality has been left out. A Visualising Display can not set the display interruptible or not interruptible, requirement R-14. The Visualising Display should support to download and run programs without any pre-session configuration. However, using Java RMI requires the Visualising Display to have local access to the class files of the application that it is to visualise. This was discovered at the end of the implementation. This can be solved by installing the required class files previous to a session or the class files can be downloaded when needed. The last solution is the best. How much work that must be done to fix this problem is difficult to estimate. A revision must be done in the future that solves this problem.

To protect the information shared in a Content Server, it is required that the Content Server is added to the collaboration instance that the content is being shared. Other protection like filtering of information and visibility of other members of the collaboration instance should be implemented in the privacy service. However, DISCOlab should have provided support for giving access to alter and control shared display sessions in DISCOlab. This feature has been left out, because of time restrictions, and is left for future work.

9.4 Simplifications in DISCOlab

Several simplifications have been done in DISCOlab. In the analysis I presented an idea having a application library with shared display applications. This simplification was done to reduce complexity of the task. In retrospect this was a good decision. To implement such a structure would be very time consuming and superimpose many new challenges that would be difficult to solve in the time frame of the project.

In relation to the application library it was also discussed if DISCOlab should have a central or peer-to-peer architecture. Centralised was chosen. For the time being, this seems like a good solution, because this lets asynchronous applications be available when no peers use them. In addition, the probability of having overloaded Content Servers in the system is low. This because each Content Server can give support for one application, having several Content Servers in UbiCollab.

The location, privacy and presence services have not been used in the project, because

the services are not fully functional at the time of implementation. The location service was under construction when the implementation of DISCOlab was finished. So was the privacy service. This made it difficult to integrate the work done in those services into DISCOlab. The location of people has therefore been hard coded and functionality has to be added to access this information at UbiCollab in the future. To create support for fetching information from the location service, should not be much work. The WebService class must add functionality to request information from the UbiCollab web service and set this in the Person and Resource classes, see figure 7.6. When a fully functional privacy service is available at UbiCollab, adaptions must be done in DISCOlab to fully utilise what the service has to offer . At the present time, the extent of this is difficult to estimate. DISCOlab is prepare to handle a working presence service.

Summary

In this chapter I have looked at the work from critical point of view. Because of limited experience in a complex field and time restrictions, the result has some limitations. However, the end product is a working prototype that give core functionality to build shared display applications on UbiCollab.

Chapter 10

Conclusion

In this project we have developed a toolkit for shared display systems on the UbiCollab platform. The work has been focused on providing an infrastructure to developers that abstracts away low-level implementation issues and that provides functionality to contextualise information in shared display systems, using the UbiCollab platform. Low-level properties that shared display systems have in common have been investigated and a set of requirements were elaborated based on scenario driven research. These requirements were used to develop a conceptual model. The conceptual model was implemented in a prototype. The prototype was used to create a demonstrator, used to evaluate the work.

10.1 Summary of contributions

During this project the following contributions have been made:

- Presented research in the field of shared display systems and toolkits for groupware.
- Presented a conceptual model for a toolkit providing support for creation of shared display systems on the UbiCollab platform.
- A prototype has been implemented, providing support to developers of shared display systems.
- The prototype has been integrated in the UbiCollab platform and its' web service.
- A demonstration has been given, illustrating how the prototype can be used to develop a shared display system.

10.2 Future work

Through the project several simplifications have been done. These simplifications are summarised in the list below. The simplifications are left for future work and can be investigated to improve the functionality of the toolkit.

- Evaluate if an application library for dynamically downloading of applications when needed by a Content Server or Visualising Display is a better solution than the present solution, see section 5.1.
- Extend the toolkit, having a subscriber list in DISCOlab, making it possible to subscribe a Visualising Display to a Content Server over several sessions, ie. the Visualising Display remembers what Content Servers the user is subscribing to, see section 5.2 requirement R-7.
- Evaluate if it will be more effective to implement dynamically plug-in downloading, instead of remote controlling the contentpane, as in todays solution, in the client and if so, improve the toolkit by implementing this solution, see section 5.4 and requirement R-11.
- Solve the class files problem in RMI, described in the evaluation, see section 9.3.
- Add functionality for protection and access control to the content shared in DISCOlab, requirement R-13.
- Add functionality to support the location service in DISCOlab, see section 9.4.
- Integrate DISCOlab with the privacy service, see section 9.4.

Bibliography

[ASGH+99] Norbert A. Streitz, Jiirg GeiBler, Torsten Holmer, Shinichi Konomi, Christian Miiller-Tomfelde, Wolfgang Reischl, Petra Rexroth, Peter Seitz, and Ralf Steinmetz. i-land: an interactive landscape for creativity and innovation. In Conference on Human Factors in Computing Systems. Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit, pages 346 – 353, Pittsburgh, Pennsylvania, United States, 1999. CHI.

[Asy] Definition of asynchronous. http://www.trainingfinder.org/cdc_lingo.htm.

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

[BBC97] P.J. Brown, J.D. Bovey, and X. Chen. Context-aware applications: From the laboratory to the marketplace. IEEE Personal Communications, 4:58–64, 1997.

[BEN+03] Julie A. Black, W. Keith Edwards, Mark W. Newman, Jana Z. Sedivy, and Trevor F. Smith. Supporting extensible public display systems with Speakeasy. In Kenton O'Hara, Mark Perry, Elisabeth Churchill, and Daniel Russel, editors, Public and Situated Displays. Social and Interactional Aspects of Shared Display Technologies, volume 2, pages 359–386. Kluwer Academic Publishers, 2003.

[BIF+04] Harry Brignull, Shahram Izadi, Geraldine Fitzpatrick, Yvonne Rogers, and Tom Rodden. The introduction of a shared interactive surface into a communal space. In Computer Supported Cooperative Work. Proceedings of the 2004 ACM conference on Computer supported cooperative work, pages 49 – 58, Chicago, Illinois, USA, 2004. ACM Press New York, NY, USA.

[Blu] Bluetooth. http://www.bluetooth.com.

[BR04] Anders Magnus Braathen and Hans Steien Rasmussen. Preserving privacy in a ubiquitous collaborative environment: Extending the UbiCollab platform. Master's thesis, Norges Teknisk-Naturvitenskapelige Universitet, 2004.

[BR05] Anders Magnus Braathen and Hans Steien Rasmussen. Preserving privacy in UbiCollab: Extending privacy support in a ubiquitous collaborative environement. Master's thesis, Norges Teknisk-Naturvitenskapelige Universitet, 2005.

[CF04] Scott Counts and Eric Fellheimer. Supporting social presence through lightweight photo sharing on and off the desktop. In Conference on Human Factors in Computing Systems. Proceedings of the SIGCHI conference on Human factors in computing systems, pages 599 – 606, Vienna, Austria, 2004.

[CNLH03] E.F. Churchill, L. Denoue Nelson, P. L., Murphy, and J. Helfman. Public and Situated Displays, Social and Interactional Aspects of Shared Display Technologies, chapter The Plasma Poster Network. Kluwer Academic Publishers, 2003.

[Cyb] CyberLink for java. http://sourceforge.net/projects/cgupnpjava/.

[DA99] Anind K. Dey and Gregory D. Abowd. Toward a Better Understanding of Context and Context-awareness. GVU Technical Report, College of Computing, Georgia Institute of Technology., 22, 1999.

[DE00] Paul Dourish and W. Keith Edwards. A tale of two toolkits: Relating infrastructure and use in flexible cscw toolkits. Computer Supported Cooperative Work (CSCW), 9(1), March 2000.

[DF04] Monica Divitini and Babak A. Farshchian. Shared displays for promoting informal cooperation: an exploratory study. In F. Darses, R. Dieng, C. Simone, and M. Zacklad, editors, 6th International Conference on the Design of Cooperative Systems - Scenario-based Design of Collaborative Systems (COOP'2004), pages 211–226, Hyeres, France, May 2004. IOS Press, Amsterdam, The Netherlands.

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

[Dou95] Paul Dourish. Developing a reflective model of collaborative systems. ACM Transactions on Computer-Human Interaction, 2(1):40–63, 1995.

[Ell99] C.A. Ellis. Workflow technology. In M. Beaudouin-Lafon, editor, Computer Supported Co-operative Work, pages 29–54. John Wiley Sons, 1999.

[Far02] B.A. Farshchian. Presence technologies for informal collaboration. In R.e. al., editor, Emerging Communication: Studies on new technologies and practices in communication. IOS Press, 2002.

[Gav92] William W. Gaver. The affordances of media spaces for collaboration. In Proceedings of the 1992 ACM conference on Computer-supported cooperative work, pages 17 – 24, Toronto, Ontario, Canada, 1992. ACM conference on Computer-supported cooperative work, ACM Press New York, NY, USA.

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

[Gra03] A. Grasso. Public and Situated Displays, Social and Interactional Aspects of Shared Display Technologies, chapter Supporting communities of practice with large screen displays. Kluwer Academic Publishers, 2003.

[Gre96] Saul Greenberg. Peepholes: Low cost awareness of ones community. In Companion Proceedings, pages 206–207, Vancouver, BC Canada, APRIL 13-18 1996. CH196.

[Gre01] Saul Greenberg. Context as a dynamic construct. HUMAN-COMPUTER INTERACTION, 16:257–268, 2001.

[Gre04] Saul Greenberg. Toolkits and interface creativity. In Invited submission to the Special Issue on Groupware, Multimedia Tools and Applications. Kluwer, 2004.

[HRS04] IM here: public instant messaging on large, shared displays for workgroup interactions, Vienna, Austria, 2004. ACM Press New York, NY, USA.

[HT04] David M. Hilbert and Jonathan Trevor. Personalizing shared ubiquitous devices. Interactions, 11(3):34 – 43, May + June 2004.

[J2M] Java 2 platform, micro edition (J2ME). http://java.sun.com/j2me/index.jsp.

[J2S] Version 1.4.2 of java 2 platform, standard edition (J2SE). http://java.sun.com/j2se/1.4.2/index.jsp.

[LH98] P. Luff and C. Heath. Mobility in collaboration. In Computer Supported Cooperative Work, pages 305–314. ACM, 1998.

[lot] Lotus corps. SameTime. http://www.lotus.com/home.nsf/welcome/ sametime.

[Ltd05] EDOX Solutions Ltd. Definition of toolkit, 2005. http://www.edoxsolutions.co.uk/glossary_of_terms.htm.

[MIEL99] Elizabeth D. Mynatt, Takep Igarashi, W Keith Edwards, and Anthony LaMarca. Flatland: new dimensions in office whiteboards. In Conference on Human Factors in Computing Systems. Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit, pages 346 – 353, Pittsburgh, Pennsylvania, United States, 1999. CHI.

[MSN] Microsoft network (MSN). http://www.msn.com/.

[NC005] MultiView: spatially faithful group video conferencing, Portland, Oregon, USA, 2005. ACM Press New York, NY, USA.

[Net] Microsoft netmeeting. http://www.microsoft.com/windows/netmeeting/.

[OPCR03a] Kenton O'Hara, Mark Perry, Elizabeth Churchill, and Daniel Russell, editors. Public and Situated Displays, Social and Interactional Aspects of Shared Display Technologies, volume 2. Kluwer Academic Publishers, 2003.

[OPCR03b] Kenton O'Hara, Mark Perry, Elizabeth Churchill, and Daniel Russell. Public and Situated Displays, Social and Interactional Aspects of Shared Display Technologies, chapter Introduction to public and situated displays. Kluwer Academic Publishers, 2003.

[QN001] Constructing a Web-based Asynchronous and Synchronous Collaboration Environment Using WebDAV and Lotus SameTime, Portland, Oregon, USA, 2001. ACM Press New York, NY, USA.

[RMIa] Java remote method invocation (Java RMI). http://java.sun.com/products/jdk/rmi/.

[RMIb] J2me rmi optional package, (RMI OP). http://java.sun.com/products/rmiop/index.jsp.

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

[SHH04] Masanori Sugimoto, Kazuhiro Hosoi, and Hiromichi Hashizume. Caretta: a system for supporting face-to-face collaboration by integrating personal and shared spaces. In Conference on Human Factors in Computing Systems. Proceedings of the SIGCHI conference on Human factors in computing systems, pages 41 – 48, Vienna, Austria, 2004. SIGCHI.

[SJ05] Børge Setså Jensen. Location-aware service for the UbiCollab platform. Master's thesis, Norges Teknisk-Naturvitenskapelige Universitet, 2005.

[SJH04] Børge Setså Jensen and Carsten Andreas Heitmann. Location-aware service for the UbiCollab platform. Master's thesis, Norges Teknisk-Naturvitenskapelige Universitet, 2004.

[SOA] Simple Object Access Protocol (SOAP). http://www.w3.org/2000/xp/group/.

[SS00] K Scribner and M.C. Stiver. Understanding SOAP: The authoritative solution. Technical report, SAMS Press, 2000.

[Syn] Definition of synchronous. http://www.trainingfinder.org/cdc_lingo.htm.

[UPn] Universal plug 'n play (UPnP). http://www.upnp.org/.

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

Appendix A

Shared display scenario

The following scenario was used as a guide in the development of a toolkit for shared display systems at the UbiCollab platform.

It is assumed that the users of the application have logged in with a username and been invited to the meeting described.

John is a project manager in a telecommunication company. Every Monday morning the team has a meeting to discuss the progress in the project and make plans for further advances. This Monday morning John wakes up half an our late and sends a message to the job that he is running late and that the meeting has to be postponed an hour, too 9.00 am.

When Sylvia gets to the office she can see John's message at the office's whiteboard, that runs a messageboard application. She can also see the reminder for the meeting telling that the meeting is scheduled in meeting room 2 for 8.00 am. Finally, an easy Monday morning she thinks for herself. She power on her laptop and starts preparing for the meeting. She is going to present her advances on the project this morning.

At 9.00 am everyone is seated in the room and Sylvia can start her presentation. She uses the presentation mode of the big whiteboard hanging on the wall. Halfway through the presentation John has a question and asks Sylvia if she mind him going back one slide. John uses his laptop to control the presentation and points at some figures in the slide asking Sylvia. Sylvia redirects the question to Mary at the other side of the table. Mary was the one designing the figures that John found questionable. Mary uses her laptop and switches the shared display to drawing mode. The display is synchronised with the other computers throughout the session. She maximises the screen so everyone can have a good look and she starts explaining while she draws. After a few minutes John is convinced and Sylvia can continue her presentation.

Mark is the next team member that is going to do a presentation. Mark is in Sweden doing some fieldwork for the project. He has connected his laptop to the shared display system the others are using. From Sweden he has been able to both look and listen to

Sylvia's presentation and the illustrations on the drawing board done by Mary. Mark also has been seeing through presence information and context information displayed in the application which colleges that have been active and contributing to the meeting. He can see that all the others, except Pepe, are gathered at the main office in Oslo. Mark starts to explain to the others how things are working out in Sweden using a headset with microphone. The others listen interested and look at the slides. They can see that Mark(in Sweden) is underlining important points with his mouse pointer.

Pepe is a Colombian consultant the company has hired for this project. He is working hand on with the building of the services the company is doing in Colombia. Before the weekend he downloaded the presentations that the others had added to the meeting. This way he can follow the presentations without being connected to the platform this morning. As the presentations proceed he can see at his mobile phone what pages and documents that are been displayed and discussed. Pepe is moving back and forth in the presentations and listens while the others speak.

After the presentations John concludes the meeting with some key points on the white-board. He gives some instructions to the members and they agree upon a todo-list for the week's work. John saves the list on the meetings folder in the server.

Appendix B

UbiCollab scenario

The following scenario guided what was going to be implemented in the prototype and what had to be simplified or left out. The scenario was used as a common link between the two groups. It is based on the scenario created in the master theses written by Schwarz, Bakkevold and Goncalves. The reason for basing it on this scenario was to show that their functionality was still provided after we implemented our enhancements to the UbiCollab platform.

In this project it is assumed that all employees are connected to the UbiCollab platform as long as some connection is available. The users are able to specify different context settings according to their availability, group belongings and inter-human relationships. The setting can be dynamically managed through different profiles.

Brian and Sylvia are both working at a telecommunication company. They are currently working on the same project, and Sylvia has some ideas which she would like feedback on from Brian. Unfortunately he is busy for the rest of the day. She decides to organize a meeting the next day.

Picking up her PDA, she opens the software client, UbiClient. She then creates a meeting. The screen asks for time, place and people. She schedules the meeting with Brian at 12 o'clock the next day at her office (room S, Telenor). She prepares some slides for the meeting, and adds the relevant files to UbiCollab.

Brian is sent an email announcing the meeting. When he starts Ubi-Client, he can see the meeting already added on the display of his PDA. He notices that Sylvia already posted some topics for discussion and some slides are available too.

On Wednesday they meet as planned. They both use their UbiCollab clients on their PDAs. When they enter the meeting room their status changes from "available" to "in meeting". Sylvia notices that the meeting room is equipped with a projector. She searches for the device with the UbiClient, the client comes up with the closest alternatives and she adds the projector to the meeting. She activates the projector, and it lights up and displays a welcome-screen, and a representation of her and Brian. She

uses the client to display the slides she has prepared on the projector. The projector shows the file on the display, her representation widget lights up and she uses the PDA to remotely control the presentation. A small snapshot of the current window shows up on her display while she taps through the remote control commands.

During their discussion Brian takes his turn of arguing. He accesses the remote control window on his PDA and jumps to some previous slides of the presentation. His representation widget lights up. But still some doubts remain about the feasibility of the project. Brian suggests that they check with John, which is an expert on that technology, and Sylvia agrees. He can see on her UbiClient that John is unavailable (picking up children in kindergarden). They still need some help and decide to try contacting Steve the projectmanager for the project. Brian uses his UbiClient to invite Steve to the meeting. At the time, Steve is in the company cafeteria, logged on with his PDA. A message pops up on his screen, asking him to join the meeting with Sylvia and Brian. He confirms.

Steve opens an audio connection to the meeting room and Brian fills him in on their problems. Steve then wants to look at the slides and searches for a shared display using his UbiCollab client. The system finds a display in the cafeteria but it is unavailable for him. He then decides to startup his laptop to use its shared display capability.

By glancing at the meeting information on her PDA, Sylvia can now see that Steve has joined thanks to a new presence widget which just showed up. Checking his available devices, she can see that he has a display application similar to the one in the projector. She says to Steve that she'll synchronize their displays, so that he can also see the current slide. She selects the two devices in UbiClient, and synchronizes them. Steve sees the slide showing up on his screen. He then examines the page and comments on Sylvia's remarks. Brian sees that the display is now being shared by all the three participants.

Steve wants to print out the slides to have a hardcopy to look at and he tells Sylvia and Brian to hold on while he does so. He uses his PDA and selects the document containing the slides, and tells the PDA to print it on the nearest printer. The PDA tells Steve which printer was chosen and shows him a map over where he is located and where the printer can be found. The PDA asks him to confirm if the printing is ok. Steve confirms and uses the map to find the printer.

At the same time Alice, which is working on a different project with Sylvia, is trying to find Sylvia. She opens her UbiClient and see that Sylvia is logged in but busy in a meeting in room S. Alice is able to get this information because Sylvia has configured her profile for Alice to be able to see her location. Alice sends a message to Sylvia telling her that she wants to meet during the afternoon for a discussion on their project. The message will be displayed on Sylvias screen once the meeting has ended.

Just when Alice is finished typing out the message to Sylvia she gets a message on her UbiClient telling her that George one of her co-workers on the project is passing by in the hallway. She opens her office door and says hi. They talk about how things

are going and then move on to more work-related discussion. From this conversation Alice gets the information she was going to query Sylvia about and she decides to postpone her meeting with Sylvia. She uses her UbiClient to cancel the message she had sent to Sylvia.

Brian asks Steve what he thinks about the idea. When Brian jumps to the previous slide using his PDA, the same thing happens on Steves display. Steve thinks it is a good idea, and he sees no counter arguments. Sylvia thanks Steve for his opinion and valuable time, and Steve leaves the meeting. His presence widget changes, and the audio connection ends.

Excited about the new idea, Brian and Sylvia continue their discussion. They still need input from John on the feasibility of the project and they tell the system to notify them once John is back in the office and available for discussion. Brian writes down a minute of the meeting in a text editor, and saves it using the UbiCollab client.

The next day, Steve is interested in reviewing what happened in the meeting. Starting UbiClient he accesses his meetings and taps the one he had with Brian and Sylvia. Tapping on the meeting notes he discovers Brian's topic minutes and reads them.

Appendix C

UbiCollab API

The UbiCollab API:

- login(String userEmail, String password): Logs in a user to the UbiCollab platform. Returns a key for the session.
- loginDevice(String deviceID, String password, String url) : Logs in a device to the UbiCollab platform. Returns a key for the session.
- createCollabInst(String sessionKey, String collabInstName, String time, String place, String notificationType): Creates a collaboration instance with the submitted attributes. Returns a identification of the collaboration instance.
- createUser(String userEmail, String password, String firstName, String lastName): Creates a user profile in the UbiCollab platform.
- registerDevice(String deviceID, String password, String name): Registers a device at the UbiCollab platform.
- addPersonsToCollabInst(String sessionKey, String[] userEmails, String collabInstID): Adds a list of persons to the collaboration instance. All persons must be registered at the UbiCollab platform.
- addResourceToCollabInst(String sessionKey, String url, String type, String friendlyName, String description, String collabInstID): Adds a resource to the collaboration instance. The resource can be a device or any other resource with an address.
- getCollabInstInfo(String sessionKey,String collabInstID): Returns a list of all information on a collaboration instance. This can be resources in the collaboration instance, persons that are members and name, place and time of the collaboration instance.
- getUserCollabData(String sessionKey, String userEmail) : Returns all collaboration instances of a user.

- getUserProfile(String sessionKey, String userEmail): Returns the profile of a user.
- getUserResources(String sessionKey, String userEmail, String searchstring, int maxResults): Returns the resources associated with a user.
- justKeepAlive(String sessionKey): Tells the platform that the entity with the given session key still is logged in.
- logout(String sessionKey, String userEmail): Logs out the user.
- removeUserFromCollabInst(String sessionKey, String userEmail, String collabInstID): Removes a given user from the collaboration instance.
- removeResourceFromCollabInst(String sessionKey, String url, String collabInstID): Removes the given resource from a collaboration instance.
- searchPeople(String sessionKey, String searchString, int maxResults): Searches for people registered in UbiCollab that fits the search string.
- searchResources(String sessionKey, String searchString, int maxResults): Searches for resources visible or registered in UbiCollab that matches the search string.
- searchResourcesClientPosition(String sessionKey, String searchString, String clientPosition, String deviceType, int maxResults): Searches for a resources based on the position of the client doing the query. The list of results is sorted from the submitted position.
- setPresenceToCollabInst(String sessionKey, String userEmail, String collabInstID, String presencePercentage): Sets the presence to a collaboration instance.
- addRelation(String sessionKey, String resourceURL, String type): Adds a relation to a resource.
- getPosition(String sessionKey, String entityID, String requestingEntityID): Returns the position of an entity in UbiCollab.

DISCOlab extension of the UbiCollab API:

- searchContentServer(String sessionKey, String searchString): Searches for a Content Server that is capable of sharing the content specified in the search string.
- addContentToContentServer(String sessionKey, String contentURL, String contentServerURL, String collabInstID): Adds the content specified to a Content Server if the Content Server can share the content type.

- removeContentFromContentServer(String sessionKey, String contentURL, String contentServerURL, String collabInstID): Removes the content specified if the content is added to the Content Server.
- subscribeToContentServer(String sessionKey, String visualisingDisplayURL, String contentServerURL, String userName, String collabInstID): Connects a Visualising Display to the specified Content Server if the Content Server is member of the collaboration instance that the user is in.
- unSubscribeToContentServer(String sessionKey, String visualisingDisplayURL): Disconnects the Visualising Display from a Content Server.

Appendix D

CD-ROM

Accompanying this report is a CD-ROM. It contains the following:

- A PDF version of this report.
- Source code for DISCOlab, the demonstrator and the UbiCollab platform.
- Javadoc for the DISCOlab toolkit.