NTNU
Innovation and Creativity

End User Service Composition

Presenting a composition tool for end users with modular architecture and a graphical user interface

Espen Nersveen

Master of Science in Informatics
Submission date: June 2007
Supervisor: Monica Divitini, IDI
Co-supervisor: Babak Farshchian, Telenor

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

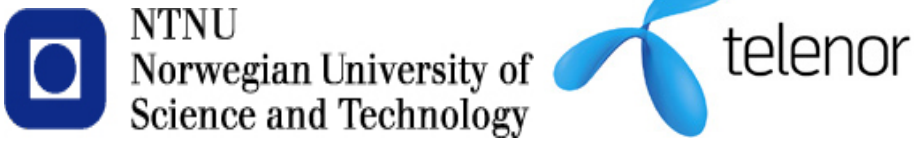
Author:
Espen Nersveen
NTNU Trondheim

End User Service Composition

Presenting a composition tool for end users
with modular architecture and a graphical user interface

May 2007

Supervisor:
Monica Divitini
NTNU Trondheim
External Supervisor:
Babak Farshchian
Telenor R&I Trondheim



Abstract

The report focuses on a possible problem to end users do to lack of control in a rapidly growing environment of computation embedded devices, and collaborative linking of various services. Such environments are often referred to as ubiquitous or pervasive computing environment. We have looked into why this problem may occur and more importantly, how to reduce the effect it may have on end users.

Our work involves the process of creating a framework that can enable end users to compose services, by connecting them in a manner that allows them to become better or more functional towards the end user then it's single components. We propose an architecture that can support rapid composition, and a user interface that can perform rapid end user service composition at any time.

From the time that a user finds the need to connect two or more services together to the user having set up a complete composition should be a task performed as quick as drawing a composition on a sheet of paper. We therefor propose a Graphical User Interface to support the end user, and we will in this report show how it is made and how it works. We will also present the architecture needed to support such a user interface.

Keywords: service composition, end user, CoPE, graphical user interface, direct manipulation, OSGi, ubiquitous computing

Preface

This is the final report for the course "IT3900 - Masteroppgave i Informatikk" (Master thesis in the course Information Technology). The structure of the course was as follows: First semester with 1 of 3 courses, introduced and defined our task. Second semester with 3 of 4 courses where we focused on problem definition and solution proposal. Final semester with 4 of 4 courses being the development, implementation and documentation.

The assignment was given by Babak Farshchian of Telenor R&I, in cooperation with Monica Divitini of NTNU.

The assignment description was given initially, but has evolved throughout this project. See appendix A for original definition.

I wish to thank Babak Farshchian, who has been my guide in this project. Relation to UbiCollab and Astra provided by Babak has been of great help while setting this project in a wider perspective. He has also been very helpful in the process of writing this report.

Also I want to thank to Frank Paaske, who has been a great help while coding example services. Without his help the current demo would not be as impressive.

Acronyms

Table 1: Acronyms

CoPE	Collaborative Pervasive Elements
DLNA	Digital Living Network Alliance
GUI	Graphical User Interface
JRE	Java Runtime Environment
OSGi	Open Services Gateway Initiative
SDP	Service Delivery Platform
UI	User Interface
UPnP	Universal Plug and Play

Terminology

Table 2: Terminology

Composed Service	A service and a relation.
Relation	A connection between two services.
Service	Any element that supports composition and/or access through tools such as CoPE is a service.
Service Composition	Any two or more services connected in any meaningful way through tools such as CoPE is a service composition.
Service Interface	Any user interface provided by the service proxy.
Service Proxy	An OSGi bundle that provides an interface to a service.
User Device	The physical device running CoPE.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

> **Chaos** *(noun). The confused, unorganized condition or mass of matter before the creation of distinct and orderly forms.*
>
> – Dictionary: Chaos

1.1 Motivation

In the recent time the notion of a ubiquitous computing environment has received a lot of attention. This is a rising trend which integrates computation into the environment, rather than having computers which are distinct objects. By embedding computation into the environment users can interact more naturally with a computerized world. [Abowd and Mynatt, 2000] for instance, shows research on the growing network of connectable components and how some aspiring technology starts to show potential. This notion is growing faster in the recent days due to a rapid development of wireless techonlogies. As embedded computation grows in the environment, the network in which they communicate is also evolving at an extreme rate. The table then turns to the user. Questions like "how can a simple user understand this vast environment?" and "how can he adapt to what surrounds him?" rise and give challenge to this new research area. This is some of the questions raised in research of pervasive computing in [Satyanarayanan, 2001], ubiquitous computing in [Weiser, 1999] or everyday computing in [Abowd and Mynatt, 2000]. An environment like this, where elements float in and out without the possibility of explicit integration, can leave end users with an overwhelmed sense of chaos.

This ubiquitous computing environment is a source of inspiration as well as frustration for anyone who wants to take advantage of it. [Newman et al., 2002] advices allowing the end user to have the control of connecting components in this environment. With this vast pool of components available to the end user, combined with the number of available connection options on each component, makes the creating of services that adopt each possible permutation an impossible mission. [Newman et al., 2002] therefor advocates the need for end users to be able to create or reconfigure these services on demand. We agree with this assessment and want to empower the end user in such environments, so that they have a better way of navigating, connecting and utilizing the various devices and services he encounters. We want to make the process from discovering to connecting to using these devices as easy as possible for the end user.

1.2 Goals

We want the end user to be able to connect any and all elements that would appear in a meaningful manner, speedy, efficiently and without having to know anything other then the fact that two or more elements *can* in fact interact. We want to give the end users a more central role while operating in a pervasive computing environment.

Developers in this environment should not have to endure a plethora of radical changes to adapt to our experimental manager.

1.3 Contributions and Project Background

Our project will hopefully be of great use to a number of projects. It is therefor necessary to keep that in mind throughout the process of design and development. We would like to see the final product as a prototype that can be adapted easily to any of these projects. If not the entirety of the prototype, then at least parts of it. We must develop highly modular code structure so parts of the code can be used in other projects.

This is a project that may be implemented in Telenor's existing SDP. This will include links to 3rd party providers and enablers as well as the internal components to handle charging, user management, content access, service ontologies, etc.

Contributions to these parts comes under the topic of "End User Service Composition". We will develop a prototype system that will serve as both a proof-of-concept as well as being a modular piece of software that can be adapted and used by the projects *Daidalos*, *Ubicollab* and *ISIS*.

1.3.1 Daidalos

Our contribution to Daidalos is research into software package management. Architecture needed to support end user management, as well as policy enforcement options for Telenor and external service providers can be used by Daidalos.

1.3.2 UbiCollab

Contributions here will be to show how an end user can become a central tool to enable their own scenarios. The definition of a user interface that efficiently interacts with the user, as well as a common link with their own architecture will be provided in this report.

1.3.3 ISIS

From our project we have granted a developer from ISIS, Frank Paaske, the opportunity to help shape our projects influence on ISIS. ISIS can use our prototype as a complete compositional framework for their own proof-of-concept.

1.4 Research Questions

- **What is End User Service Composition and why do we need it?** We want to make clear what it is and why it's needed in the growing ubiquitous computing environment. What makes the end user the right tool for the job?
- **What is needed to facilitate End User Service Composition in terms of**
  - **Architecture?** What architecture will best suit rapid composition by the end user? How can we build this architecture to to best suit highly mobile and demanding end users?

– **User Interface?** What is the best way of presenting the user interface? What does the user need in his user interface and what do we need to design to facilitate it?

1.5 Research Method

This project started with the general motivation of enhancing the end users ability to use and connect elements in a ubiquitous computing environments.

Our research method is illustrated in figure 1.1. We started with a deeper description of the problem to better see what was needed in the longer run. From there we investigated state-of-the-art with relations to our problem and developed a solution proposal. These three parts where collated and sent to an international review board for the conference *Interact 2007* http://tuim.inf.puc rio.br/interact2007/home.php. The review from the board allowed us valuable insight to form the solution proposal somewhat different.

From the revised solution proposal came the design and development stage. This stage was divided into two parallel processes: Architecture, which is the platform, and Graphical User Interface, for end users to use the platform. Though the two processes were separate in nature they were developed synchronously. Ideas have been formed in the user interface, realized in the architecture and then extracted and used back in the user interface. The process have been iterative with regards to the development cycle of both processes. To develop the graphical user interface we used low-fidelity user testing to give us more iterations. The end result was a functional prototype tool to enable end user service composition (CoPE)[1] [2].

[1] **CoPE** - (**Co**llaborative **P**ervasive **E**lements)

[2] **Cope** (verb) - To face and deal with problems, or difficulties, esp. successfully or in a calm or adequate manner

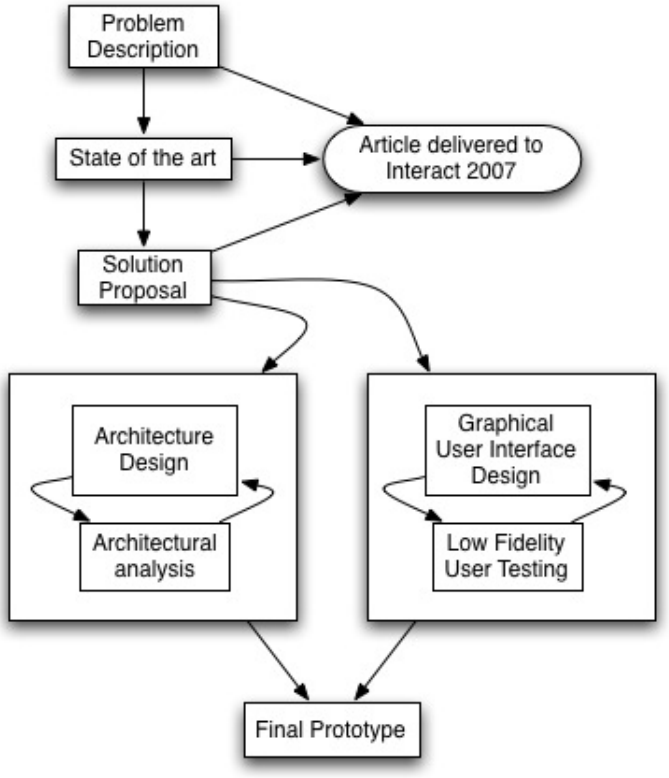


Figure 1.1: Research Method

1.6 Report Outline

Chapter 2 contains the **Problem Description**. Here we will go deeper into the reason and motivation for doing this project, why this project is needed, and what we intend to look into and solve in later chapters.

Chapter 3 is the **State of the art** exploration, where we reference work previously accomplished to show where in the line of research we are placing this project. We will here introduce more the idea of empowering end users.

Chapter 4 presents our **Solution Proposal**. This is what we are basing prototype development on. Key features needed and provided are presented and discussed.

Chapter 5 will introduce our implemented **Architecture** in greater detail. This is the background system needed to present the user with a viable tool for service composition. Relation between our prototype and the ubiquitous computing environment is the main part of this chapter.

Chapter 6 gives an overview of the implemented **Graphical User Interface**. This is the end result of the process from selecting what type of user interface we wanted to use, to the enrichment of end user experience.

Chapter 7 will summarize the **Implementation** from Chapter 5 and 6 into a briefer overview of what our prototype can do, how it relates to the user and how it relates to the related environment.

Chapter 8 and 9 finally gives the overall **Evaluation and Conclusion** followed by the **Future Work** needed to ensure the survivability and viability of this project, as well as showing some future potential.

1.7 Key Concepts and Scenario

Before we delve into this report it's important to establish what we aim at when we are talking about service composition. In the first part of this chapter we directed our focus on end users in a ubiquitous computing environment. We have chosen to call all elements within these environments: *services*. We define a service as anything that is capable of supporting the collaborative end user management of our approach. Any elements categorized as being a part of the ubiquitous or pervasive computing genre are services. Examples are devices which supports input or output, like a video projector, a DVD player or a printer. Services can also be information providers, like flight schedule, or an Electronic Program Guide (EPG).

> *Any network enabled element that supports composition and/or access through tools such as CoPE is a service.*

A service can be connected to other services. This is in the nature of the service and we will call a service that is connected to other services a *composed service*. A composed service consists of the service of origin and the connection. The connection itself is called a *relation*.

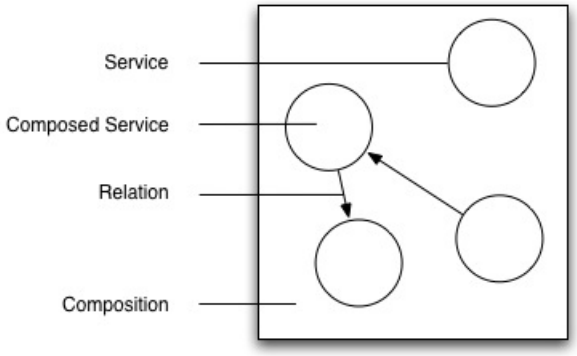


Figure 1.2: Definitions

> *When connecting a service to another service we call the connection a relation. A service with a relation is called a composed service.*

When one or more composed services are connected, we define the compilation as a *service composition*. A service composition is a combination of collaborating services and/or resources that together obtain higher functionality than that of it's parts. It can be seen as a composition by either a user's perspective or the system's perspective. Such a composition can also originate from other places, but in this article we will refer to service compositions as only those compositions made by or can be reconfigured by tools like CoPE.

> *A group of composed services is called a service composition.*

1.7.1 Scenario

This scenario describes a day of our user, Bob. Bob has just acquired a video projector and a brand new sound system to put in his living room. As he is finished unpacking and wants to test it, he sits down in his couch and pulls out his PDA which include our software. The first thing he does is to discover his own DVD player. As he acquires the control of the DVD player, it shows that it can connect to an audio and a video output device. He locates the two new devices from a list that searches nearby devices and connects them both to his DVD player using CoPE. He connects the sound to the speakers and the video to his projector. Bob has now made a service

composition comprised of three composed services. He chooses to save his composition. Now anytime Bob wants to watch a movie in his living room he can start this service composition and have it act just the way he wants.

Another day Bob is at his friends house. He and his friends get together to watch their favorite soccer team play. Before the match starts, Bob is playing with his composition from the other day and discovers that that setup also works in his friends house, as he has a DVD player, a TV and a couple of speakers. Bob asks his friend if he wants Bob's setup to adapt to his setup, and of course he answers "yes". Bob selects his saved setup and chooses to share it. CoPE makes a shareable version of it and a list of PDAs to share it with appears. He chooses his friends name, and he recives the composition. As he opens it it shows the setup similar to what Bob had, but every item is exchanged with a description of what is needed. He selects each service in turn and links them with his own services which he finds in the same manner as Bob did the other day. However, it goes a lot faster because the search of the devices to fill in are narrowed considerably when the connections are already made. Bob and his friends continue to watch the match. Bob's friend continues to use this setup after the evening soccer match.

Chapter 2

Problem description

A problem clearly stated is a problem half solved.

– Dorothea Brande

2.1 Services and Control

In the last chapter we introduced the end user and the ubiquitous computing environment. We also defined services, as the components found within the environment. A service could be composed or atomic. An atomic service provides a set functionality to the end user and works as a standalone service, but we want to focus on the composed service in this project. A composed service provides means to reach out to other services and collaborate. It can therefor become more useful then what it was before it was composed. By collaborating with other services the new service composition can perform with greater accuracy, more functionality, with better quality or in a more suited manner towards the end user.

We believe the end user to be the key to controlling ubiquitous computing environments. Our task is to enable the end user's ability to compose services, while still being able to interact with them in the same manner as he would normally do.

The usability aspect of composing services is the main focus of our project. This means the user's level of control in a ubiquitous computing environment, both in taking advantage of service functionality as well as the con-

nection abilities of one service towards other services. We define the end user's level of control related to services in five points:

- **Discovering** - The process of searching and installing controllers for a service.
- **Connecting** - The process of composing services.
- **Starting / Stopping** - Allowing the user to turn on and off the installed controllers for a service.
- **Using** - Providing the user with a user interface to each installed controllers, from where the user can control the service's properties.
- **Sharing** - Sharing compositions between devices running composition tools like CoPE.

If all points on this list are supported in the user interface and considered easy by a user, we say the user has a high level of control, but if the user has a hard time any of this points, the entire overview that he may have had is quickly lost. We will argue that if any of these points are presented to the end user in such a way that he finds it hard to accomplish a task, the entire feeling of control is lost.

As we mentioned earlier, the volume of services available to the end user is rapidly increasing. In figure 2.1 we illustrate the rising number of services at any given time available to the end user by the blue line. If the current trends are kept at status quo, we will most likely see a corresponding decrease of the user's level of control, represented by the red graph in the same figure. This will in turn render the intended multifunctional and highly interconnectable services virtually useless when the user does not know how to properly take advantage of them.

The design of services in these environments is quite challenging. Therefor, research into giving more control to end users have been a rising theme. Normally developing new collaborating possibilities in software engineering includes the stages illustrated in figure 2.2. First an expert identifies the need for a service connecting to another service and defines what it should do. Next a developer implements a service that corresponds to the expert's demands, automating the collaborative process, with connections and connection methods. Finally the end user have the completed new service

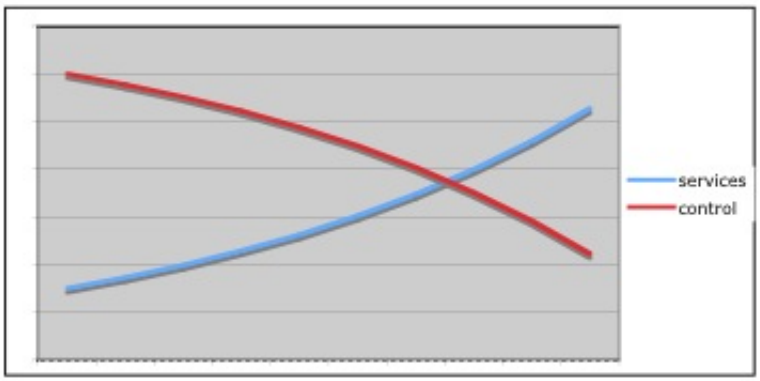


Figure 2.1: Number of Services vs User Control

collaboration. The ubiquitous computing environment however needs an alternate approach. [Crabtree et al., 2003] talk about anticipative design. He urges developers to enable more communication possibilities then that of which the developed service is intended for. By allowing this the user has more freedom to create own relations that can perform in a different or new manner. To reach our goal of enhancing end user control we need to make them more central in the process of designing, implementing and using services. Our research is focused on the tools available to the end user in such an environment and will follow and supplement several other researchers. [Davidoff et al., 2006] presents the need for end user involvement because the user will feel in control over an otherwise uncontrollable environment. [Dey et al., 2004] states that end users should be the ones that decide what to create, and in which circumstances, because they are in fact the ones with the most knowledge about what they usually do in the course of a day. The notion is also supported by [Gajos, 2001].

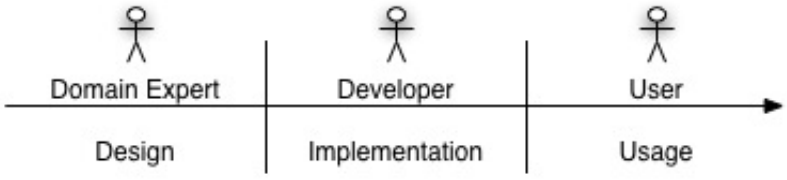


Figure 2.2: Problem with Conventional Service Development Process

2.2 Control vs Automation

To increase the level of control we must give more options of control to the user, which is not as straight forward as one might be inclined to think. It is important to notice the possible issues of increasing the control. If one is to choose a degree by the question of wether or not we want full control or automation, we are faced with having to sacrifice the one side we don't want (fig 2.3). The choice of Linux or Mac OS X, is a good example (although Linux is becoming more and more user friendly and OS X has the controllable UNIX at it's base). Linux offers it's users complete control with it's open source software, while OS X offers the efficient and easy to use graphical user interface. However, Linux command prompt is definitively not easy to use for a user who is not intimately familiar with it's commands, and with OS X it's virtually impossible to change options that are not directly implemented in the user interface.

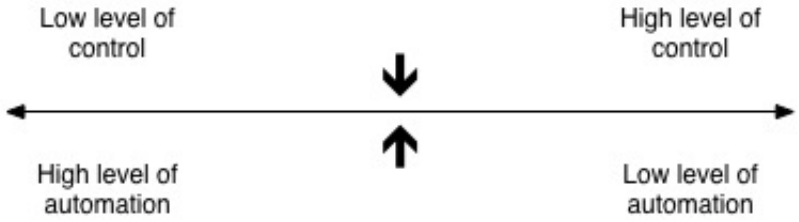


Figure 2.3: User Control vs Automation

Both ends of the line have positive and negative aspects to it. A high degree of automation does a lot of work on behalf of the user. It will also make the inherently technical processes transparent to enable the user to focus on the task at hand rather then how the task needs to be performed. The increased automation and transparency can however cause a general concern for misfit of services. Services that demand rapid adaptation to various changes have to manage most of this on it's own, which will leave the user clueless if he did not expect the service to behave in a certain manner. Additionally the service might even behave wrongly.

On the other hand, one can go for high level of control. This will reduce the problem of misfit in changing services, as the user has total control over the actions of each service. This will increase the service's potential as the

possibilities become more or less endless. But "with great power comes great responsibility". The user has the options at his hand. As the options increase, so does the complexity of the tasks. Faced with insurmountable permutations of possibilities, the user has to have a steady mind whenever he wants to do something. This is a strain on any person.

As both ends of the scale presents good as well as bad qualities, we are faced with the decision of which way to go. We want the user to have more control in a rapidly changing environment, but we don't want to impose strenuous management options. We must attempt to find the "best of both worlds" so to say. We are aiming for the center of the arrow, but we also want to find a way to achieve a higher level of control as well as a not excluding automation. That means (from figure 2.3) dragging the arrow pointing downward towards the right end, and the arrow pointing upward towards the left end. Other articles on the theme of user control vs technology control are [Davidoff et al., 2006], [Lee et al., 2006].

2.3 Rapid Context Change

In these times, end users rely more and more on technology to cope with rapid context change. That is; the services adapting to changes caused by the user's context or to changes in the service itself. In a highly mobile user, context change comes often, and adapting the context change to software decisions becomes more challenging. Research into context adaptation in ubiquitous environments have been conducted by, amongst others, [Abowd et al., 1997], [Gajos et al., 2002], [Salber et al., 1999] and [Wang et al., 2004]. When the environment changes from each setting a user accesses it, context adaptive management is needed. We believe giving the end user more control will be the best method for making the environment considerably easier to use.

The main cause of context change comes from increasing mobility of the user. More devices are made to be handled on-the-go. We will have to decide which way is the most effective way of handling the increased mobility of the user. For instance, [Abowd et al., 1997], [Helal et al., 2005], [Holmquist et al., 2001] uses positioning and proximity to enable context adaptive service access and composition. We want the user to take control over his own mobility. Proximity and location could be one of the guiding options. We

should in the end provide mobility support by reuse and adaptation of service compositions.

Automation must be an option in the prototype which can further be developed with reasoning and decision making tools, and even artificial intelligence. The user should have options on what he wants to do at any time, but he must also have an easy way of automating certain options he uses often. So in the event of his choosing, a certain flow of information dictates an event on a selected service. The prototype has to support this in a way that makes the creating and managing of automated fall naturally to the end user.

2.3.1 Learning to use CoPE

Because of the insurmountable obstacles that may eventually appear if the environment outgrows the ability to keep control of it, the notion of serendipity is presented [Newman et al., 2002], [Humble et al., 2003]. Serendipity in the field of end user service composition means the end user performing a certain task, and while performing discovers possibilities he was not familiar with before. Playing around with compositions should be made so that the user can "learn **while** doing", and not "learn **before** doing".

Chapter 3

State of the art

If I have seen further it is by standing on ye shoulders of Giants.

– Sir Isaac Newton

3.1 Introduction

This document will present some variations of user interfaces that exist, or are being developed, that could support the notion of end user service composition. We will evaluate the different variations from our project's perspective.

3.2 Choosing the User Interface

Following is a brief description of various methods of user interaction, along with some work that has been done on each point. We have defined some key points that we need in our project. The points are illustrated in figure 3.1.

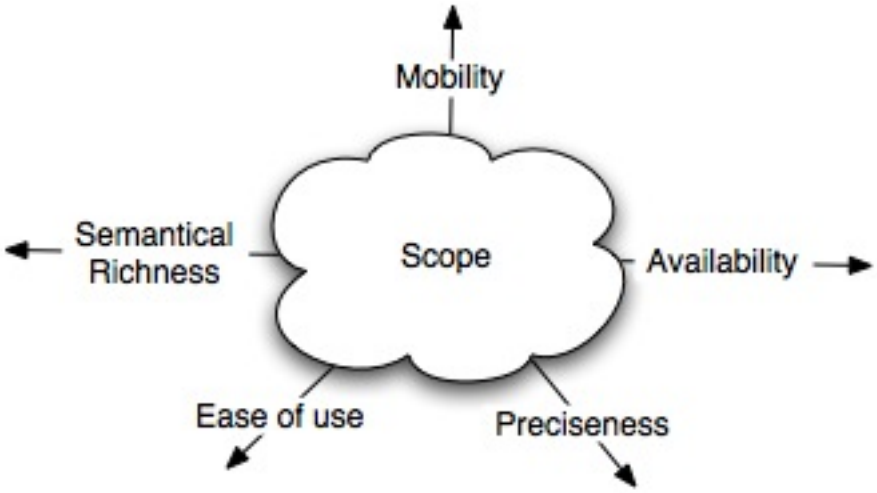


Figure 3.1: Scope Directions

- **Mobility** - How easy does this method work with a mobile user.
- **Availability** - How well can this method hold grounds in todays market.
- **Preciseness** - How precise can the user control this method, and how well does the method respond to the user's input.
- **Ease of use** - How easy and efficient is this method in end user service composition.
- **Semantical Richness** - How versatile is the interaction with the end user.

We give points ranging from 0 to 4 on each, with 4 being the highest and 2 representing neutral. A neutral score means it does not excel nor fall behind, and is not a reason for choosing or dismissing it. We will go with the one which yields the overall highest score. The score table is shown in figure 3.2. We will look into input and output in the following variations of user interfaces:

- **Textual** - Pure text.
- **Manipulative text** - Same as textual, but with more abilities, like drop-down menus or auto-completion.
- **Vocal** - End users speak commands to the system. The system responds by speech.
- **Tangible** - Physical objects representing actions on a service.
- **Graphical** - Virtual approximation of tangible interfaces, using icons and drawing in a 2d environment.

3.2.1 Textual

Textual input using a keyboard, is the basis of UNIX systems and gives high control to skilled users. This is an easy method of relaying commands to a system, and a good point to start. Command lines can be sequenced in scripts, that form a line of connections to enable the system to do a specific task. This method has a low fault rate, because the commands has to be exact to work. An incomplete command does nothing, or performs wrongly. This however draws heavy on the users working memory as well as long term memory as the user must remember each connection command, as well as each element he wants to connect. We also feel that feedback from this user interface is not understood fast enough for an end user. Notably this scores highest on *preciseness* and *semantical richness*, but looses out on *ease of use*.

3.2.2 Manipulative text

Newman et al. [2002] proposed this method, through a PDA and a web browser. The concept is simple, and the functionality is clear. They also performed some low-fidelity tests with direct manipulation, but rejected this due to real estate cost and complexity towards the end user. The availability of this method is very high in this list, as every computer has a web browser,

and more and more PDAs and mobile phones are shipping with browsers integrated. There is no need for extra software to make this method work. All you need is the web browser. It scores high on *mobility* because it can almost always be used in any situation. To use it you only need one hand free, and it can be implemented on static devices as well, so this method gets the highest score on *availability*.

3.2.3 Voice

Voice commands can be compared to scripting. Instead of written commands you can speak the commands. Work on this method of interaction is performed by, amongst others, Gajos et al. [2002], in the article *End User Empowerment in Human Centered Pervasive Computing*. They introduce their agent *Alfred*, which acts on behalf of the system as a parser and conveyer of the spoken commands. The user will talk to the system as if it were a person, and ask it to perform tasks. By implementing the agent to respond and ask like a human would, the user can feel more at ease with the system. The system was made to record macros. The macros would perform tasks in a manner defined by the user. The macros can be compared to service composition.

The most positive aspect of voice is the fact that it has a high value of availability. With exception of mutes, every body has their voice with them at all times. Another great effect from voice interfaces is that the user does not need to use his hands, nor even eyes, at any time. A user is free to perform other tasks and still not impair the systems functionality.

A general drawback of this is the fact that voice recognition is difficult to implement successfully. Most people speak in different ways, and the system needs to learn each persons voice characteristics. This does not seem to be a problem in Gajos et al. [2002], due to their background agents *ReBa and Rascal* as well as the extensive vocal feedback provided. Some of the features they present is at the very core of our problem also. It would not be an unlikely mode of interaction in our project, but we choose not to implement it because we want the user to have the ability to use our systems composition service in places such as meetings and other public places where loud speaking is kept to a minimum. As for comparing with scripting, this also has the drawback of users having to remember commands. This gets the highest score on *mobility*, but looses out on *ease of use* and *preciseness*.

With a good artificial intelligence reasoner to analyze input and output, *semantical richness* can also rate good.

3.2.4 Tangible UI

Connection between artifacts in a ubiquitous computing environment is proposed in Holmquist et al. [2001], as a very physical interaction. A user would move artifacts in close proximity and physically shake them to have them connect to each other. While our system also supports proximity as a context variable, it is not feasible for our system to use the shaking method as a means to connect two artifacts.

The connection between elements that are not in the immediate vicinity is also a necessity, so if this was to be implemented it would have to be alongside some other method. However such research as put out by Svanaes and Verplank [2000] is highly interesting as to how users relate to tangible interfaces. For instance, the *iButton* shows some good points as to how the user relates to context and migration of data. *Availability* and *mobility* rates lowest in this method. A tangible UI can be excellent on stationary services, but not really viable as a UI for mobile use. Most places will not have the same tangible computation UI available. *Semantical richness* and *ease of use* however can rate high because the user is able to "feel" what he is doing. And depending on the implementation, the feedback can be very intuitive.

3.2.5 Graphical

This is the version where the user will have a pen and a touchscreen. The objective of this is direct manipulation of graphical objects that represent the user's environment. Humble et al. [2003] presents a graphical user interface that uses jigsaw puzzles as a metaphor for connections. The composition is represented by a "solved" puzzle. This is a great method for serendipitous service composition, as the user does not need to remember each object in his environment, thus he is able to keep his mind focused on the composition itself. This method rates highest on *ease of use* because it is, in our minds, the quickest form of interaction. *Mobility* rates high, because it could potentially run on every new user device. *Preciseness* also rates high because the user can get all types of feedback from the UI efficiently.

3.3 Summary of Scores

Here follows an overview of how we rated each method of interaction. Green squares are the highest rated, which we think has the best options for what it is rated on. Red squares are the ones we feel lack something in what it is rated on.

	Semantical Richness	Ease of use	Preciseness	Availability	Mobility
Textual	4	0	4	2	2
Manipulative text	2	2	3	4	3
Vocal	3	1	0	2	4
Tangible	3	4	2	0	0
Graphical	2	4	4	3	4

Figure 3.2: Scoreboard

Chapter 4

Solution Proposal

The best way to escape from a problem is to solve it.

– Alan Saporta

4.1 Introduction

This chapter will introduce our solution proposal for a software prototype called CoPE. We will first choose the user interface that we feel suits our project best, and provide some reasons for choosing it. Then we will discuss what is needed from our architecture to support the user interface, as well as present some central concepts that needs to be in place for the architecture to work.

4.2 Proposed User Interface

Newman et al. [2002] used a PDA with a touchscreen and a pen, but rejected the idea of direct manipulation, claiming it would take too much real-estate from the screen on a mobile device. It is true that directly manipulative objects in a graphical user interface requires a lot of "unused" graphical space to not overflow the user with information. Humble et al. [2003] however showed that this does not have to be true. With the screen sizes and resolutions of todays mobile smart phones and PDAs, the available space on a smaller screen should suite our project rather well. Our prototype will use directly manipulative objects, that are dragged and dropped. This can also be used with a stylus on a touchscreen.

4.2.1 Availability

One of the main reasons for selecting the touchscreen over other methods is it's availability. The trend has mostly been that more high ranking businessmen and enthusiasts have been the key consumer of devices with touchscreen, but that trend is changing. More and more smartphones and PDAs are becoming affordable, and with more people using such devices, as well as the consumer having a wider range of devices to choose from, we feel that it could be a viable method for this project.

4.2.2 Direct Manipulation

The touchscreen will provide the necessary tool for enabling direct manipulation. This will enhance the ability of usage from novice users, as they do not require to learn complex syntax to understand the system. Through use of metaphors we hope to make it even easier. The use of metaphors in the design of directly manipulative interfaces is very important as it is easily recognized by the user and does not put unnecessary strain on his working memory. Madsen [1994], Marx [1994]. Feedback from a graphical user interface also have a better way of communicating with the end user, as it is not restricted to text only. Illustrations can be used to optimize the feedback for better understanding.

4.3 Final Thoughts on Chosen User Interface

We will not profess the superiority of the graphical user interfaces, and direct manipulation over the other alternatives presented. We have seen, from the articles referenced, that they all bring something to the table, so to speak. We will however state that we feel comfortable with the choice, and believe it will fulfill the needs of our project as best we can see at this point in time. Most importantly we believe that this method will provide us with the necessary tool to facilitate serendipitous service composition by the end user.

We propose a set of low-fidelity user testing in an iterative user interface design process. Since only one are going to be developing, we will use low-fidelity tests, such as Wizard-of-Oz testing ([Dahlback et al., 1993]), to increase the number of iterations. This will give us a broader perspective and better secure the way for future versions.

- We will design a graphical user interface.
  - Using low-fidelity user tests.
  - Using iterative design and development phases.

4.4 Mission statement

The "Original Assignment Description" (Appendix A) page 61 is the assignment in raw form from what we described before the start of the project. This still applies, although some modifications has been made through the course of the project. We will define these on the following paragraphs.

> A generic and common framework is needed for user interface "pieces" to be defined, combined, used etc.
>
> A generic tool is needed to set up and configure user interface pieces for the current applications.

It is important to know that we are not talking about combining two service interfaces into one, and thereby generating a new interface. We are talking about designing the background system and a user interface that will allow the services to efficiently give the end user the ability to connect services as well as configuring them to his preferences.

In the end the user must not be micromanaged. We can not make a wizard like program for connecting and managing services, as this method is not versatile enough to accommodate all the services we have considered that it should be able to support. It certainly can not handle all the services we might encounter in the future. The prototype must give the user the full freedom he would expect from such a tool, while still giving him guidance without it being a nuisance. A Graphical User Interface that uses direct manipulation on objects representing services should give the necessary freedom.

The direct manipulation gives the user freedom to shape his surroundings. To guide we need a central matchmaking system that will quickly display hints or guides to a user when he is performing on services. The hints must come in the form of feedback to the user on what he is allowed and not allowed to do. This matchmaking process must be highly tuned for rapid response, and will be explained better in 5.8 Ontology.

Ubicollab follows a Service Oriented Architecture (SOA). We propose building our structure in a strict Model View Controller (MVC) architecture to cut down development time and still maintain an easy way of converting to SOA in future work. We also propose to use technology from OSGi (Open Source Gateway Initiative) in our implementation.

> **OSGi technology is the dynamic module system for Java**
> *OSGi technology provides a service-oriented, component-based environment for developers and offers standardized ways to manage the software lifecycle. These capabilities greatly increase the value of a wide range of computers and devices that use the Java platform.*

OSGi gives us a plug-and-play platform designed for service management. We will from this build a control point for end users. Services are are OSGi "bundles". The bundles are wrappers around pieces of code which makes possible our collaboration methods.

- We will produce a generic and common framework for defining, combining and using Services.
  - Use and adapt existing theorems and structure from UbiCollab.
  - Implement prototype on OSGi platform.
  - The prototype must show how it can adapt to the changing environment of ubiquitous computing.
- We will produce a generic tool available to the end user to set up and configure Services.
  - Define ideal way of presenting the user interface.
  - Give the user guidance without micromanagement through simple matchmaking.
  - Implement prototype with functional example services.

4.5 Services

The most ideal way to test and implement prototype services would be to take advantage of existing services. Unfortunately the services we can utilize today are not built to take advantage of anything else then a fixed number of other services. This is a piece of the puzzle of ubiquitous computing that

still needs fitting. We must not expect services to change and therefor design the structure of "service handling" to enable existing services to be adapted without changing the services themselves, rather then proposing redesigns to the services in question.

Work has also been done by projects like UbiCollab on this subject. Our prototype must use the same concepts in order to be applicable in future implementations. We will have to adapt the existing theorems so that we clearly show what is needed to cooperate with the related projects. Most central part of the theories is the concept of a *Service Proxy*. A service proxy is an adapter for services that incorporates methods for services that enables them to work with Ubicollab and CoPE. It is a piece of software that exists between the actual service and our software. The services themselves are stationary, and the service proxy is handed out by the service (or a service repository) with means of communicating with the service. This will allow our project to "handle" service proxies which in turn handles it's parent service. We propose implementing service proxies, and adapting them to suit the user's compositional needs.

- We will design for existing services.
  - Define the process of adapting existing services.
  - Implement example services.
  - Refine the process for future work.
  - Define what is handled by CoPE and what is handled by Services.

Chapter 5

Architecture

Design is not just what it looks like and feels like. Design is how it works.

– Steve Jobs

5.1 Introduction

This chapter will go deeper into the software parts of the prototype. We present the theories we have used and how they are implemented. CoPE and it's relation to services are the key elements here. The main parts of the architecture are illustrated in figure 5.1, where 2 services are installed, 1 composition has been made, and we have connected to an external discovery and ontology. Our prototype exists within the OSGi environment and consists of these main parts:

- **DiscoveryManager** - Handles discovery and installationof services through external discovery.
- **OntologyManager** - Handles matchmaking between services through external ontology.
- **ServiceRegistry** - Internal registry for the installed services with methods for managment (start/stop/uninstall).
- **CompositionManager** - Handles creating, storing and activation of service composition of installed services.
- **CoPE** - The interface to CoPE from a User Interface.

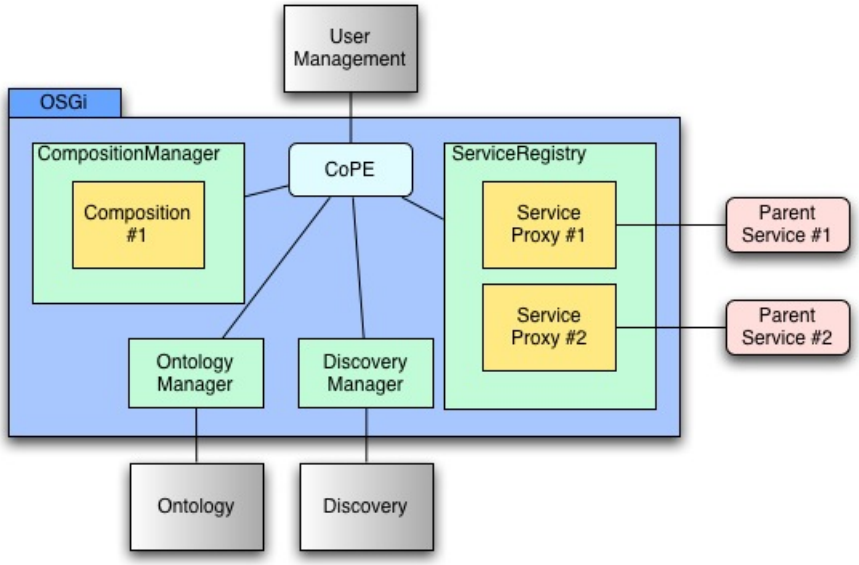


Figure 5.1: CoPE and it's relations

5.2 CoPE and Services

As mentioned, the architecture needs to fully support plug-and-play services. CoPE needs to know what services are installed as well as how to use them. This is not a big problem when using the OSGi framework, as it comes with several needed functions that supports necessary management of the installed services. The problem, however comes when you are in need of adding support beyond that of simple management. Services that are natively designed to run on it's own should not be hindered in running as it would outside CoPE, but with added management options the services can become more then what they were initially designed for.

The simple solution would be to make CoPE tell any installed service to start, and have that service pop up any interface it chooses to run. This would work straight out of the box with OSGi, but it lacks the ability to communicate with other OSGi bundles because it is as capsulated as any other physical bundle in its environment. It would have to be built with static methods to communicate with other bundles, as well as with CoPE.

This could in turn be solved by simply allowing CoPE to be a mitigator between bundles, and have it give bundles the ability to locate other bundles. This however, causes more problems then it solves. Every service now has to know every function of every bundle in order for a system of meaningful communication to work. This can become infinitely complex when more and more services are added to CoPE, and even more services are added to CoPE's ability to discover them. In order to allow the services to perform whatever task they need to complete and still let CoPE be a manager of services rather then a mitigator, a new way of thinking is required.

The solution to the problem is given partly by the related project UbiCollab (1.3.2). The idea was to separate core functionality and service functionality. We have designed CoPE to handle proxy services instead of entire services. The plan is that all functionality related to CoPE and managing of services are embedded in the proxy service, and any actual running of service functionality is handled by the proxy service and does not need communication with CoPE. This allows CoPE (and thereby the user) to have complete control over how the service is handled, while not putting restraints on the functionality of the service itself. A service runs as it would, both inside and outside CoPE.

5.3 Service Registry

The central part of CoPE is built around a service registry. The service registry is the core of CoPE's communication opportunities. All service proxies are listed in detail, containing the methods for using the services. An illustration of how the services relate to CoPE, via service proxies is shown in figure 5.1, where the service registry contains three installed service proxies. The service proxies in turn communicate with and manages their parent service.

5.4 Modularity in relation to Ubicollab

Because of the fact that the prototype should be implemented in other projects it is important to keep it modular. The most modular structure related to UbiCollab is SOA (Service Oriented Architecture), and the implementation should support this structure. We will develop the prototype with a great emphasis on a strict Model-View-Controller architecture. This will allow the code to be easily translated into SOA when the time comes.

The reason for not developing for SOA at once is that it is much easier to do testing locally than distributed. This will yield better result in our prototype as we get more time on our hands. The structure of communication in the developed SOA environment of Ubicollab (and the other projects) are also not entirely completed, so we wanted to have the code running locally before adapting it to other projects.

The main part of the prototype must be the part related to composition. Both the framework for enabling composition structure as well as the basis for adapting the framework for other projects has to by fulfilled. The solution will have to show how UbiCollab can utilize this project. What parts are in focus, and what interfaces are available for them. The main part associated with Ubicollab is sharing of compositions. When a user has made a composition he should be able to quickly share it with friends. We will develop a subscriber/subscription solution, where a user can subscribe to a composition. This will be developed in MVC rather then SOA, to improve production time, but still retain SOA abilities.

5.5 Functionality and APIs in Services and Service Proxies

The proxy services need to support all functionality necessary for the end user to complete his task of managing service compositions. The user has a sense of interacting with the services even though everything CoPE interacts with is the proxy service. For CoPE, very little knowledge of the actual service is needed. The bare essential however is that we know the actual address of the service. This must be conveyed through it's proxy service. The reason being that the method of connecting devices needs the service addresses. If we were to use the service proxy's address, then all communication would have to go through the device running CoPE. This will put a bottleneck on the communication, as well as forcing every proxy service to handle both Ontology searches as well as discovery. We chose to eliminate it by saying a proxy service has to announce the address of it's service (fig 5.2).

As the figure shows the simplicity of the latter illustration is quite a compelling argument for announcing the original service address. We can then tell the proxy service to tell the service to connect to a given address. We

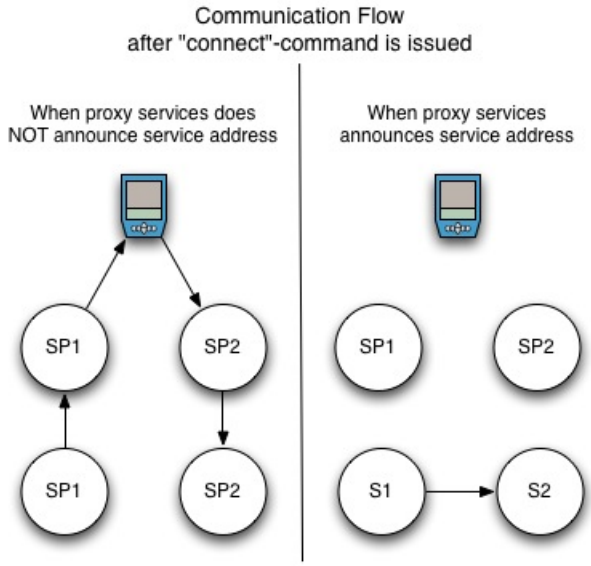


Figure 5.2: Data Flow after Relation is added

will then be able to reduce the proxy service's complexity requirements drastically. By applying ontology to our local service registry and composition manager, we can be sure of what we are telling the services to connect to will actually work, and the services can perform the connection task without having to worry about whether or not the services are compatible. The service connections are stored locally within CoPE's composition manager. In the initial figure 5.1 we have a composition manager with one composition and a service registry with two service proxies. This is what the internal representation would be if one is to make a composition. The composition contains two composed services that is linked with the service proxy, and a relation that is linked with both composed services.

5.6 Handling a Service's User Interface

One of the main problems of running services from within CoPE, and thereby within the JRE, is that the functionality the proxy service provides are not easily transferred into CoPE without CoPE knowing their structure in advance. Implementations of specific classes that provide standardized methods for a proxy service to use is a must, and this in turn will put

restraints on any service developer. This problem appeared clear as we were designing the methods for sharing the user interface of the services with CoPE.

The first proposal to solve this was to have the service deliver a Java class that CoPE could open in it's own environment. We found this to be unreasonably hard, as well as restraining service developers too much, and therefor opted to go in a different direction. The original proposal was to have CoPE ask for the service's interface, the service serializing it's interface class and sending it to CoPE, and then having CoPE opening the service class in it's own environment. We decided to walk away from this option as every method that the service wants to use (which originates from the service interface), CoPE needs prior knowledge about, unless we decide to build a communication module that a service proxy can use. This will again restrain possible communication between the service interface and the service proxy.

Earlier research in our project suggested moving a service's user interface by means of a descriptive file that could be parsed and converted to a functional GUI on a receiving end. We used the technology Thinlet http://www.thinlet.com/ to achieve this, but found that we could not use this method on the grounds that we would then have to force service proxies to use this method.

We have from this arrived at the decision to give CoPE the ability to ask services to initiate their interface, instead of asking them to give it away. The service specific interfaces would then open as a standalone interface with no knowledge of CoPE. Having the service interface open outside CoPE does not mean that CoPE has less control over the service. It's only difference is that the service interface is run parallel to CoPE, and not within. The same amount of control and communication to and from CoPE is still applicable to the service proxy, but by using this solution we have eliminated a possible bottleneck of communication from service interface to service proxy. The service proxy is also free to open it's service interface by other means then CoPE's implemented Java SWING approach.

5.7 Composition

The main part of discussions have been around the topic of the actual composition of services. In a first solution proposal we described a three-level

composition architecture that we felt, at the time, gave us the needed functionality to facilitate proper end user service composition. As we continued the development of the prototype we discovered that the intended three-level composition was not the optimal solution. The structure was sound and would probably work well in a more controlled, but we are developing mainly for the average Joe end user. We therefor needed to revise our approach. We will present the original three-level architecture and the revised architecture in the following subsections. Both the original and the revised architecture reflects on to the scenario described in 1.7.1.

5.7.1 First Proposal - 3 level Composition Architecture

Composition Template First stage of a composition. A composition template is the most generic description of a composed service. In this stage a user will define the characteristics of every element that is to be a part of the composed service. It will also contain information of relations between elements and data management descriptions. Various inputs and outputs can also be connected at this stage as the template contains enough information about each object. Templates is the only shareable version of a programmed service and can be copied and passed along to different clients. The template is stored as an XML file with knowledge specific to the selected generic service description. When a template has been completed, a user may choose to distribute it to other users so that they can share the same setup. E.g. in a distributed meeting, with people attending from another location, they can download the predefined composition and set the empty service descriptions to be his or her own service. For instance, in a template one can define a video output device, and in the meeting each of the attending can set the display to be his own PDA. If a user decides to start from this stage he will start from scratch. He will have a greater sense of the structure of the composition, but loose some speed compared to starting from other stages. This also gives the user the ability to pre-define a service composition when he has the time. He does not need any intimate knowledge of any of the services he is planning on using, only how he wants to use them.

In the scenario from chapter **??** the composition template comes into play when Bob wants to share his composition with his friend. He could not share his entire setup from where he left of because that setup had instantiated elements, meaning it was fitted to his house. When Bob chose to share it, CoPE located the underlying structure of each element into "a device

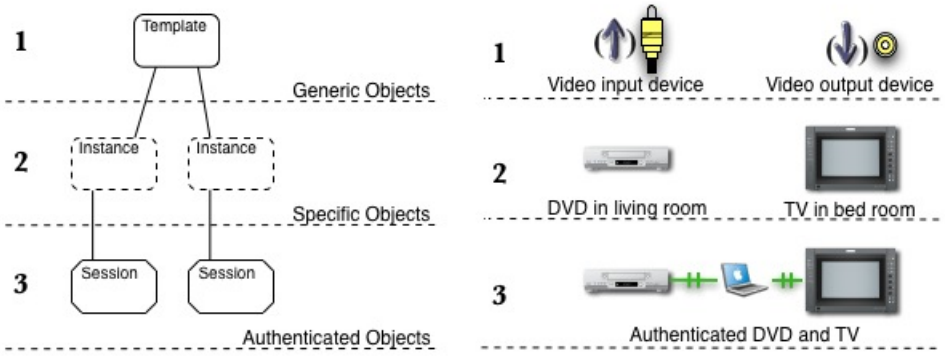


Figure 5.3: Composition Process - 1. Template, 2. Instance, 3. Session

capable of video and audio output, connected to a device capable of video input and a device capable of audio input". This description suited both Bob's house as well as his friend's.

Composition Instance A composition can be instantiated from a template, where a resource is allocated to each service. In this stage the elements contained in the template will be instantiated by locating and addressing a specific physical element that corresponds with the criteria of the template element. An instance is typically stored as a copy of the Composition Template file, with additional information about which specific element each element corresponds to. There can be many instances of the same Composition Template that is adapted to different environments and specific elements (see fig5.3). In the first scenario, Bob made a composition instance for his house starting from this stage, and when he shared it with his friend, his friend made an instance of his own. They both had the same underlying template that supported both houses.

A user can also create a composed service starting at this stage. He could locate objects around himself without creating a template first by simply searching for them and dragging them onto his workspace. An instance can also be decomposed into a more generic service template if the user decides to make the service from this stage. No processing is required as every object in a service composition has a template file.

The quality of available connection is included here, but only if the user expressively says so. E.g. a user may have a preference to what quality an audio connection has to have. He might find a DVD player which supports surround, but no audio output device that does. In this case the user should be able to get this information before he initiates a session. If the user does not state filters for the connection, two objects should always use the best possible connection unless a specific is identified by the user. This will be handled on a layer below ours, and will be implemented in future releases.

Composition Session Third stage of a composition. This is the final stage before having made a Composed Service. In this stage information about each element is shared and collected through the network made by the composition. Handshakes and authentication through Discovery and Security Services are processed. A composition session corresponds one-to-one with a composition instance, but can be adapted to another instance if the user decomposes it to a template and then makes another instance. We decided to support this in a final layer of composition to reduce the need for sharing sensitive information over any network before it is actually needed. Sessions could later support such technologies as web browser cookies so that it remembers most information needed for the handshaking process.

The user can, as all other stages, choose to start composition here. He may want to activate a single service, only to discover later that it will work better with other services connected. We did not want the user to have to rebuild an already working service or service composition. Our theory is that a user will be inspired to test connections and service creation without making a session, and that this will provide great efficiency as well as increasing the confidence of the user while composing services. This will be tested once a functional prototype is completed. We do not yet know which stage will be preferred to users, but we believe each stage is necessary to give the user enough freedom. As Newman et.al. discovered, the users had several different preferences that their team did not foresee. Some preferred semi-completed templates and others wanted something more like making the service from scratch. Some located elements nearby, as others searched by room number. We therefor want the user to be able to choose which stage he wants to start from, so that no users are left behind.

5.7.2 Revised Composition Architecture

The original three level architecture provides some necessary functionality, but we have in the process of implementing our prototype found better ways of achieving the same thing.

While the three level architecture gave users an optional starting point at the Template, we found through low fidelity user testing that none chose to start from this level. When asked they stated that it was much easier to start at the bottom. We therefor decided to rely more on instant service discovery. We state that the user will use discovery more actively then we originally anticipated. The user also expects an easy to use discovery to perform much more to their liking then if they were to use Templates. We will still enable distribution of compositions, and the system will respond and act in the same way. By allowing service discovery to discover remote services, the user can still make service compositions without being in the same room as the needed services reside.

Most functionality related to describing capabilities of a service has been moved to an ontology manager and distributed along with the service proxies. Basically it means that both the process of discovering and installing, as well as connecting services makes decisions based on matchmaking processes of a distributed matchmaking system. The ontology of our service matchmaking is described more thoroughly in 5.8 Ontology.

The other reason for having a Composition Template was for sharing purposes. The template is intended to provide us with sharing a generic description of a service composition to other devices, and then have them search for matches to the generic descriptions contained. The generic description is now accompanied with every service proxy. When a user wants to share a composition, CoPE will generate an XML file with the descriptions from the service proxies in the composition and the links between them. The template is therefor no longer an option for the end user, but a tool CoPE can use to provide sharing of compositions.

We have also given the prototype the ability to share compositions in another way. We now give a description of the composition with addresses to the service proxies. This makes the sharing process much simpler for the receiving device as well as the transmitting. The transmitting device already has the addresses in it's service registry, and the installation of

a service proxy (the OSGi bundle) is natively supported within the OSGi environment. This way the receiving device gets a description containing the installation paths of the service proxies, with the corresponding relations. When they are processed they become composed services on the receiving end. The received composition is then identical to the the original. This should allow for more rapid sharing of local compositions, such as when a person wants to share a composition he has made for a specific location.

5.8 Ontology

The ontology part of our project is perhaps the most abstract part. We have enabled CoPE to handle ontology searches and ontology bases from a distributed perspective. The problems arise when the user wants to match services against each other. We need this to give the user feedback without micromanaging his decisions. The way of doing this is to match services in real time whenever the user starts to create a connection. Each time he does so, and for each service he has in his composition, we must give accurate feedback on wether the services match or not.

An entire ontology and methods to search within it is not possible to have running on any type of portable device today. It will simply take to long time for a match query to give efficient feedback to the end user. We therefor had to distribute the processing power to a number of levels. Mainly this includes three locations: A service proxy repository running on a fixed server, the CoPE running on the end user's device, and distributed matchmaking services running in the user's available network.

To increase effectivity in searches we can for example process matchmaking already before we know what services the user wants to use. By doing hashing on this level we can utilize clusters and other "supercomputers" to do the matchmaking for us. This will leave us with a repository that knows all matches and mismatches, and can tell this to the user when he requests a service proxy. We must then rely on a central database for matches between services. The user also needs to give information about what services he currently is running with the request for the new service. The repository can then get the matches for the requested service and send that with the response. In this way the user gets the matches with his new service proxy.

When the service proxy containing the matching information has arrived at the user device, the second level of matching takes place. The matches and mismatches is stored in CoPE for every service proxy that is installed. Matches are collected from whatever form we chose to implement an ontology in. This is the highest level of hashing that is possible. CoPE stores only the simplest values and matching queries so that the process of matching is exceptionally quick, and appears instantaneous to the end user.

If the first and second level of matchmaking is possible to achieve then the third location is not expressively needed. The third location is a backup solution if the previous solution does not work. If we choose to implement this, then we would want to have a central database for matchmaking, but we do not need a central repository. We would also need the description following service proxies to contain a unique identifier for itself that can be mapped one-to-one with a service in the matchmaking database. The request sent by the user device will then travel to the matchmaker and return quickly with the matches and mismatches. This values are then stored when the user installs a service proxy.

By implementing the ontology manager in the proposed manner we will only see an increase in the time it takes to install a service proxy. The user will have instant feedback when it comes to the actual composition. The thing that is implemented is CoPE's local storage of matches and mismatches, which will enable future work on ontologies to be quickly implemented. The rest of the process is related to future work and therefor not implemented and tested.

5.9 What we get from OSGi

OSGi provides some highly needed management functions for plug and play services. We have chosen OSGi for exactly this reason. As described in State of the art, OSGi bundles Java classes into manageable plugins for the OSGi platform. Every service proxy is bundled like this, and we use the standard methods provided to install, uninstall, start and stop service proxies. They can be installed from a URL and updated with the click of a button. This management can also be done remotely, if the user gives the access for it. We have the ability to control what a remote session is and isn't allowed to do. The OSGi framework basically provides a usable service registry, with life cycle model view of each service (see figure 5.4).

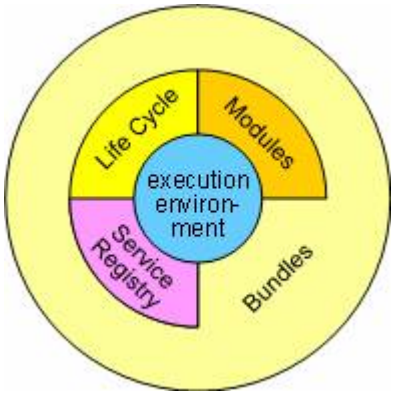


Figure 5.4: OSGi Framework

OSGi is built as a software container, meaning that software installed (in our case service proxies) has some functionality integrated that allows for efficient transmitting of service definitions and possible connections. We have opened the option of having service proxies deliver such content to both discovery and ontology managers. By doing this we facilitate distribution of the matchmaking process, as described previously in 5.8 Ontology.

5.10 Storing Sessions

As we mentioned in the solution proposal, we wanted to be able to store a session, as it was surly needed if the user would be able to turn off his device and continue using his services without reinstalling them. This presented us with a number of synchronization issues. We decided to put some controls into the service proxies, so that they could register themselves after an eventual crash, or after reinstalling them from outside CoPE. This also allowed CoPE to stop without saving it's registry. All services now implement a listener, that listens for CoPE registering in OSGi. When CoPE is loaded, restarted or reinstalled it will tell OSGi that it is ready for action and OSGi will from here tell all service proxies that they can reregister with CoPE.

The reregistering process is a very light weight process, so the services (nor CoPE) uses any more time then they would if they were to store and load the service registry from a file. Almost every part of the original registering process has already been performed, and works as well after the reregistering. The simple process of retrieving the name from the service proxies is the only

thing needed for CoPE to have the same operating functionality as before the restart.

5.11 Security in Service Proxies

Each service proxy gets a subfolder in CoPE's execution folder. This can be used by the service proxy to store variables of it's service, or any other variable it sees fit to store. Any items the proxy wants to store should be stored and accessed in this folder. Currently there are no restrictions implemented, so this is a measure to be dealt with in the further development. The design of the system is currently that a service proxy has a subfolder with the same name as the server in "CoPE/services/[name of service]", which it can use. CoPE can access this at any time, so nothing can be hidden from the user. It should be an easy task to add security features to this folder.

In future work this should be implemented more securely in some form of a "preferences manager", on which the user stores variables that the service proxy can read from. Only the user should be allowed to write to the disk. Service Proxy should only be allowed to read.

Chapter 6

Graphical User Interface

Everything should be made as simple as possible, but not simpler.

– Albert Einstein

6.1 Introduction

In this chapter we will give an overview of the current edition of our graphical user interface. We have delved into the process of designing the selected *Graphical* user interface. Through this chapter we will not refer to parent and proxy services, but to services in total. This is because the user should have the illusion of handling services rather then representations of them. Related allusion is the TV and the TV remote. If asked what users do when changing a channel, they will most likely reply that they "change the channel on the TV", they do not "act on the remote".

6.2 Design Process

We needed to user to be comfortable with the GUI, and we therefor used an iterative design process with a couple of low-fidelity user testing. The process contains 4 major revisions described in the following section. Illustrations and screenshots of the iterations are listed in Appendix B.

6.2.1 Iteration 1

In the first iteration we had initially thought that CoPE should be a quickly accessible tool within a web portal environment, resembling more

the solution of [**?**]. The web portal solution was short lived, and we therefor had to extract the tool into a standalone environment. The next image, iteration 1b, contains the same tool taken out of it's portal environment. We knew at this point that more functionality had to be formed within the composition tool.

We also proposed having a "master service" as the main point in any composition. The master service would always dictate how related services should behave. If there were more then one "source" service then that service would have to be a master service in another "set" which in our current terminology is a composition. If they were to be connected they would connect between the two sets. The communication between sets proved to difficult to implement. Also we found that the user might be flooded with sets if he were to actively use this version.

6.2.2 Iteration 2

Iteration 2 proposed the first devision of actions. Discovery for finding and installing services, Favorites for storing compositions, Sandbox for making and managing a composition and Device to manage the currently selected service's interface. The Device section was meant to contain the interface of a selected service. In the architecture chapter we discussed how CoPE relates to service interfaces, and here is the reason for no using the Device section. We are opening the interface outside CoPE's boundaries to enable the interface to also work without CoPE.

We also proposed that the user could select how many parts he wanted to show at any given time. For instance, in iteration 2a, sandbox, discovery and favorites is shown, and in 2b sandbox and discovery are shown. At this stage we applied user testing to see if they thought the devision and management of the GUI parts were good enough. The response was clear: The users found the screen to contain more information then needed at any time. They were not able to manage what parts were open as efficiently as we hoped. Instead they ended up having all windows open at all time thereby giving them more unnecessary then necessary information.

The Sandbox was also designed for free manipulation of services. A service could be selected and deselected. When it was selected a button would appear below it that when pushed activated the creation of a relation with the source being the selected service. The user testing showed us that this

was not the ideal way to go. Users had trouble finding our what to do in the first place, as well as the fact that we noticed way to much time were spent clicking on services.

6.2.3 Iteration 3

This iteration, illustrated in iteration 3, was concentrated on the Sandbox. We wanted to first of all remove the connection button from the service. By removing the button we could also reduce the clicking frequency of the user. We wanted to make the user drag connections in the same way that he would move the service itself. It would be the next iteration that would solve this problem.

6.2.4 Iteration 4

The following sections will deal with what we have created in the final prototype. The iteration 4a and 4b are screenshots taken from currently working scenarios. The scenarios are described in chapter 7 under Working Scenarios 7.3.

6.3 Discovery, My Services and Favorites

We have structured the user interface in such a manner that the user himself can decide the level of simultaneous interaction with services, by separating actions the user can perform into four main parts. "Sandbox", "Discovery", "Favorites" and "My Services". This was a thought from iteration number 2 B after we decided to go for a separation of portal and CoPE and refined when we wanted to open service interfaces outside CoPE.

Our focus on this project is contained mostly within the Sandbox. This is the part where service compositions and usage occur. Following is a brief description of Discovery, My Services, and Favorites. After describing what is needed and preferred from these parts we can begin a deeper description of our Sandbox.

6.3.1 Discovery

Discovery will focus on all things related to discovering and installing services. This will have the entirety of the process from a service being available to the system, to the point when it's available to the end user.

From an architectural point of view, this means finding the a service the end user searches for, getting it's description and installing the proxy service. For the end user this will look like installing a driver for the desired service. After installation, the service will appear in My Services.

6.3.2 My Services

My Services provides administrative options to the user. If a service should be stopped, started, or uninstalled, this is where it is done. Currently this is only a simple list, but should in future feature a rich environment for navigating installed services. From this point a user can add a service to the Sandbox. He can also open a service interface without any connections being made. Some services accessed and used by CoPE do not necessarily have to connect to other services.

6.3.3 Favorites

After a composition has been made it is stored in favorites. In the next section we will describe our Sandbox, where the actual service composition takes place. We wanted to have a separate storage room for compositions. Automation of composition loading as described earlier can be implemented here. The sorting and navigating of stored compositions should also support location based filtering, for automation purposes. A user should be able to have the active composition change when a context change occurs.

6.4 Sandbox

We want the user to become familiar with his surroundings, and if he hesitates and does not know exactly what he needs he should be encouraged to try things without having to "call someone to ask for help". We therefor use theses metaphors to help us in the development: **"Touch your surroundings"**. A user can physically touch objects on the screen. **"Feel the connections"**. The system will let you know if a connection is possible or not. Through the use of ontology descriptions, CoPE disables connection options that are not possible. With drag-drop functionality, the user can make a connection like he would drag a physical cable from a service to another. **"Play with compositions"**. The most essential metaphor. Learn by trying, play to get better.

The sandbox contains two important elements: **Services** and **Relations**. In iteration 3 we defined what we wanted the final Sandbox to look like in a minimalist form. The final product screenshots of iteration 4 is in figure B.6 and B.7.

6.4.1 Services

A service is shown as an icon, which tags along the service proxy when the user installs a service, and the name of the service directly below. The user is free to move the service around as he wants within the Sandbox. An imagined use of the free positioning of services is that a user can organize services within the Sandbox as they would appear physically in the real world. For instance, a TV service and a DVD player service could be located north (in front of the user) and room lighting service can be located relative to the location of a lamp in the room. This way a user can activate "room compositions" that looks like the current setup of the physical room.

6.4.2 Relations

Different services can of course have different methods of implementing the actual connection towards other services, and we do not intend the services to change any of those methods, such as communication protocols and physical links. We do however want this to be semi transparent to the end user. We have implemented a GUI option that can turn off showing any difference at all between different connection types. With the option used, the connection is a connection as far as the user is concerned. Default is to show what connection type a given relation is comprised of, but it should be up to the user to decide on what advanced functions are shown at any time. We should also implement an alternative that does not show relations, so that when the user is finished composing he can start using without relations cluttering his view.

Relations are handled by the user in the same manner as a service: Drag and drop. To eliminate the need of an extra button below the service to enable connecting (like we had in iteration 2) we decided to go with a "socket" on each service in the Sandbox. If the user starts a dragging motion within this socket, the service itself stays stationary and a line appears that can be dragged to the desired destination. Each time a user starts a relation creating process, we match all services in the Sandbox to the originating service. This is illustrated in figure B.6. When a user drags a line from an

example service "Weather Service", it is matched with the other services and as we can see; that service can only connect to "Sound Service".

The user is also able to choose *when* he wants the composition to activate. He might want to play around with connections while another composition containing the same services are active. By disabling "Automatically Initiate" the user can play with virtual relations rather then real relations. When this option is deactivated the services are not told that a relation has been made. It is only when the user chooses it that the services are connected in the real world. This way the user can play with the system, even if the composition in some way are not able to initiate.

6.5 Multiple User Interface Support

One of the great features of designing the system by an MVC-pattern is that the system and the GUI can run independently. The GUI uses the underlying architecture as a model, and performs task on and listens to changes in the model. This ensures that a single line of code is all that is needed to add a GUI to the system. That line of code initiates the GUI and passes the respective models along with it.

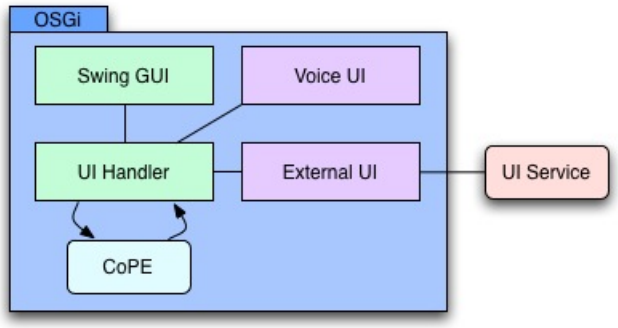


Figure 6.1: UI Management

The GUI can from this handle the model in whatever way it wants. This has allowed some additional features for further development. Though the GUI is currently a Java Swing implementation there is no limitations in the code that does not allow for other types of UIs to work just as well. For

instance, an implementation of a voice activated user interface would work just as well. It can even complement the already existing Swing interface, as they both will work on the underlying MVC architecture. When a change is made by one interface, the change is made on the model, and thereby automatically updated in all interfaces, depending on each implementation. This will allow a user to have a freedom to choose how and where he wants the interface.

Any services can also be built to support this feature. By implementing a handler for a connection of type "interface", it can set it's own supported variables for displaying or announcing vocally the interface on other devices then the user's device. We will allow the services to choose this by themselves rather then enforcing the option upon them. The underlying model also gives the interfaces methods of gathering the capabilities of the user device, so that it can tailor it's interface to fit.

Another interesting thing to notice is that one can also perform such feats as delivering the various interfaces to remote services. For example, a GUI can be shown on another device then the device running CoPE. By simply adding a new line of code invoking an adapter to a remote interface, changes to the models can be done from other physical locations. The user will by this gain freedom to move between rooms, and have the desired interface shown on any number of other physical input/output devices with an implementation of the same adapter. This can in turn be linked with other context data to provide the user with seamless interface roaming when entering rooms that supports better devices to show interfaces then that of his own device.

Chapter 7

Implementation

An acre of performance is worth a whole world of promise.

– William Dean Howells

7.1 Current implementation

7.1.1 CoPE

CoPE is implemented in Java 6.0, but is tested and compatible as far back as Java 1.4.7. The OSGi framework is version R4, with bundle versions 2.0, but CoPE can also run on versions as far back as RC3. Both the CoPE platform and GUI are running stable.

7.1.2 User Device

CoPE has been tested on laptops and stationary personal computers only. It has been tested with a variety of Java and OSGi versions, so it can be implemented once a mobile device with those versions are obtained. The current operating systems used are Linux and Windows. Both stable.

7.1.3 License

The entire code structure of CoPE is licensed under Apache License [Apache].

7.2 Example Services

We have created some example services in this project that support connections in various ways. The following services and connection abilities (with service proxies) have been created:

Weather Service

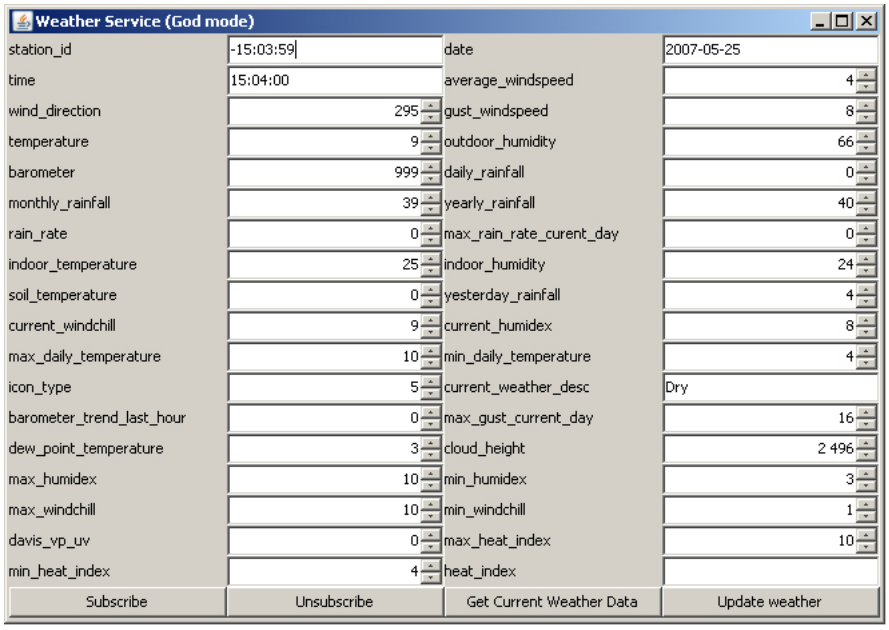
- Two outputs
  - Alarm
  - Weather data
- 

Figure 7.1: User Interface of Weather Service

Weather Service is a service connected to a remote database which contains various weather data. The data is pulled from the database and into our service. From here it sends weather data to any services supporting weather data input.

Sound Service

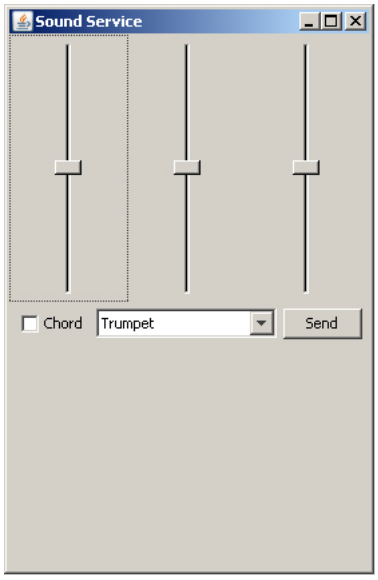
- Many inputs
  - One note
  - Three notes
  - Three notes as a chord
  - A music pattern string
  - Weather data
- 

Figure 7.2: User Interface of Sound Service

Sound Service is a listening service. It is designed to take a broad specter of inputs and convert them to notes or chords. Depending on what the input is, the resulting "music" can be quite fun and weird.

Color Service

- Two inputs
  - RGB
  - Weather data

Color Service is also a listening service, designed as the music service. It takes a broad input range and converts any input to a color which it can show on any service proxy belonging to itself.

Alarm Service

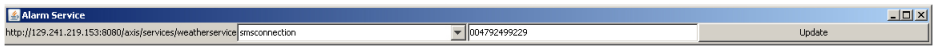
- One input
  - Alarm
- Two outputs
  - SMS
  - Email
- 

Figure 7.3: User Interface of Alarm Service

Alarm Service is meant to be a future generic service that takes critical input from any device that wants to support an alarm function, and relays the alarm to any other output device that can feature an alarm function. More on this function follows.

SMS Service

- One input
  - SMS

SMS Service is a simple service in terms of what the user gets to play with. It is designed take short text as input and send it as an SMS. The interface of the proxy contains the option of which number the user wants to send the message to. All input is handled by CoPE's drag and drop.

Voice Service

- One input
  - Text

Voice Service takes a simple text input and converts it to speech.

7.3 Working Relations

Our ontology manager matches each service the user installs to each service he already has installed, and the user can therefor be guided at any point of the creation of a composition. The descriptions of connection abilities is gathered from the service proxy and stored in the local matching lists for quick access. Possible relations so far are the following:

The weather service may send data to the alarm service, sound service and color service. The weather service is our primary source of data in our scenarios. This has the raw weather data at it's core, but also supports tailoring to other services. It is not designed specifically for color and sound services, but the color and sound takes such a wide range of inputs that it works any way.

The weather service also has a specific alarm output. This is meant to be a generic output / input description for future versions. The alarm service is the only service that supports the alarm input, but it supports most known output services and relays alarms to these if it is connected to them. In the illustration, the alarm service is connected to two output unit; the SMS service, and the Voice Service. So when an alarm event occurs the alarm is spoken in words through the user's device as well as an SMS being sent to a mobile phone.

7.4 Working Scenario

A user can install any service proxy related to any service we have listed in the "Discovery" section of CoPE. These are updated automatically from their respective services. The services can connect in the manner described in "Working Relations", and the user will only be allowed to connect them as such.

The user can install, uninstall, start and stop a service in the "My Services" section of CoPE.

In the "Sandbox" section, the user can move services freely as well as create and remove relations. He can create any relation described above in "Working Relations".

When opening the interface of Weather Service a user can see all data coming from the weather station. When opening Sound Service he can select which musical instrument he wants the output to perform in. When opening the SMS Service ha can choose what number he wants to send SMS to.

Chapter 8

Evaluation and conclusion

Wisdom is found only in truth.

– Johann Wolfgang Von Goethe

8.1 Evaluation

Evaluation has been provided in a number of ways, and we will list them as follows:

Daidalos has given valuable input on the task of service proxy management. With the Daidalos project came the need to handle bundles in an OSGi environment. The solution of this was given as a management tool for end users that would support access rights on a login function. The tool developed here has been adapted to our project and is used in the same manner as proposed to Daidalos.

ISIS has given the most input on the art of service composition from an end user perspective. All example services are derived from some part of ISIS scenarios. ISIS is still in the process of starting up, so the example services are completely made from scratch, to support migration to the ISIS project. Evaluation in relation to this project has been very good. The method of direct manipulation in a graphical user environment has been well recieved, and CoPE will most likely be used in some form in the ISIS project.

Interact 2007 gave us good feedback on our proposed architecture and graphical user interface. We found that the structure was indeed sound, but needed some fine tuning. We have addressed these issues in our report and intend on writing an updated article after the end of the course.

UbiCollab is the closest related project. From Ubicollab we found some architectural properties that we had to build upon. The Service Proxy proposal and service registry both link directly to this project. We have also seen that the structure and proposed architecture of the Service Discovery relates to CoPE in the manner we intended at the very start of our project.

8.2 Conclusion

Our problem described a possible lack of user control in a future ubiquitous computing environment. We proposed giving the user more control in the form of leaving more decisions in the hands of the end user, by the means of end user service composition. By implementing in Java and OSGi we have shown that the software can handle multiple services simultaneously, in a plug and play manner, on any platform. The user has control of each service as he would have normally. In addition, our solution gives each service a standardized way of presenting the end user with a meaningful way of creating relations between them. Making service proxies for each service has shown that we can give the user control without impairing the services functions. In fact, the service might even improve relations to other services.

Handing composition over to the end user meant that we had to put matchmaking processes in the user's software, removing all things related to service discovery and matching from the services. This presented new problems to how we were to handle matching processes. To solve this problem we have shown that an internal list of matches, that can be populated in a number of ways through ontology services, gives the necessary feedback to the end user when he is composing. Rapid matchmaking gives the user instant insight into what is and isn't possible.

Finally we have shown the entire concept come to life through our Graphical User Interface. We have shown that though the area of available space on a mobile device has a limited amount of pixels, our solution of direct manipulation is still a valid method of presenting composition to the end user. We have also shown how direct manipulation of service objects can

give the user a virtual environment that reflects the user's own allusions to how services are positioned and how they act on each other. By allowing the end user to store different compositions for different settings, he himself can decide when and where he wants them to activate.

We stated from the start, and still believe, that giving end users the ability to perform service composition is the way to go in the future. Having end users perform services composition will give a better controllable structure of collaboration between services, and in the end may even increase the mobility and collaboration abilities of the services themselves.

Chapter 9

Future Work

The best way to predict the future is to create it.

– Peter F. Drucker

9.1 Architecture

Firstly we must convert the code structure from MVC to SOA. This will make the software better suited to be used in multiple projects, both at Telenor and NTNU. When complete it will be used as a resource for handling composition of generic Java Objects that in turn can be comprised of whatever the project requires. We will implement a subscription based structure that will allow multiple sources to manage a single composition.

Parallel to making SOA architecture is a different task: To make the matchmaking process a reality we require a functional ontology and search functions within that ontology. The ontology has to implement our Java Interface for ontology managers, but otherwise can be built freely.

We also want to exchange our discovery manager with UbiCollab's discovery project. This should work very well with CoPE as they both work on the same principles on service registry and service proxies.

9.2 Graphical User Interface

We presented the idea of making the Sandbox in our program implement drawing functionality. We want to make this a reality for future versions,

giving the user the ability to draw physical objects that can better illustrate and show how compositions relate to the real world. In the further away future, we may even see the implementation of 3d mapping of the physical world within CoPE.

With drawing in Sandbox, we can also implement other methods of navigating the ubiquitous computing environments. We want to see an implementation of zooming and physical navigation from location to location. For instance, when a user is in the living room, the living room composition is active. If the user wants to activate a service located in the neighboring room he could zoom from a "room-view" to a "floor-view" and back into a "room-view" of the kitchen. The physical location of services can give a more intuitive way of navigation. Again, in the further away future, we may also here see the implementation of 3d mapping, to better illustrate the environment.

Appendix A

Original Assignment Description

This task will be done in the context of two European project called Daidalos and ASTRA. User interfaces to computing environments are changing rapidly with the advent of mobile and ubiquitous computing. Some of the technical challenges to be handled are.

- A generic and common framework is needed for user interface "pieces" to be defined, combined, used etc.
- A generic tool is needed to set up and configure user interface pieces for the current applications.
- A framework and API for service providers to be able to discover, recognize and deploy pieces of user interface that are available to the user of the services.
- The user interface in a ubiquitous environment might be speech-based, physical, haptic and so on. This task will investigate the above issues, and will explore further issues not foreseen in the task definition. The scope of the task is the following:
  - Provide a set of benchmark scenarios for user interfaces to pervasive and ubiquitous interfaces.
  - Provide a state-of-the-art survey of user interfaces to pervasive and ubiquitous computing.
  - Develop architecture and design for a service-oriented framework for specifying, developing and deploying ubiquitous interfaces to network services, and develop a prototype of this.

Appendix B

Graphical User Interface Iterations

Following is illustrations and screenshots of the GUI design process. Each iteration is described in detail in 6.2.

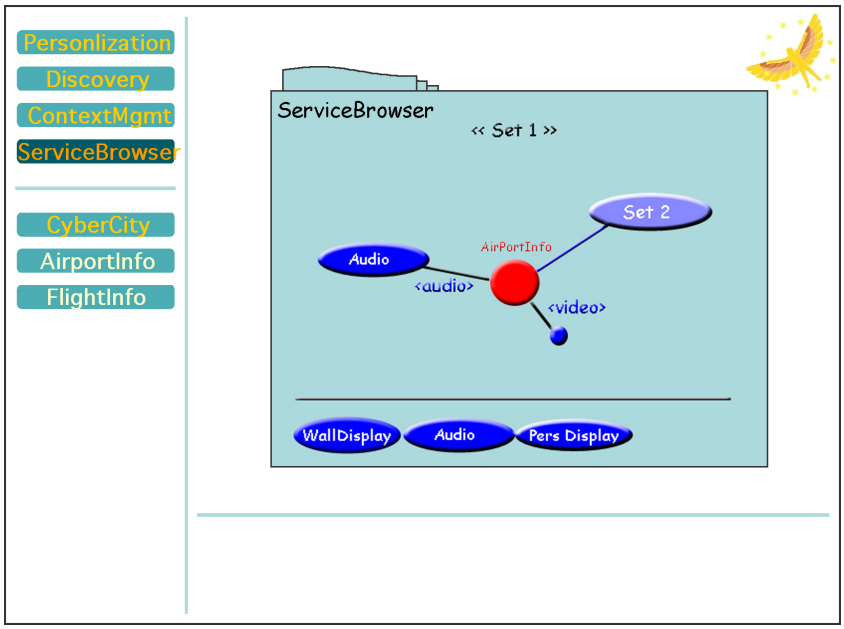


Figure B.1: GUI iteration 1a

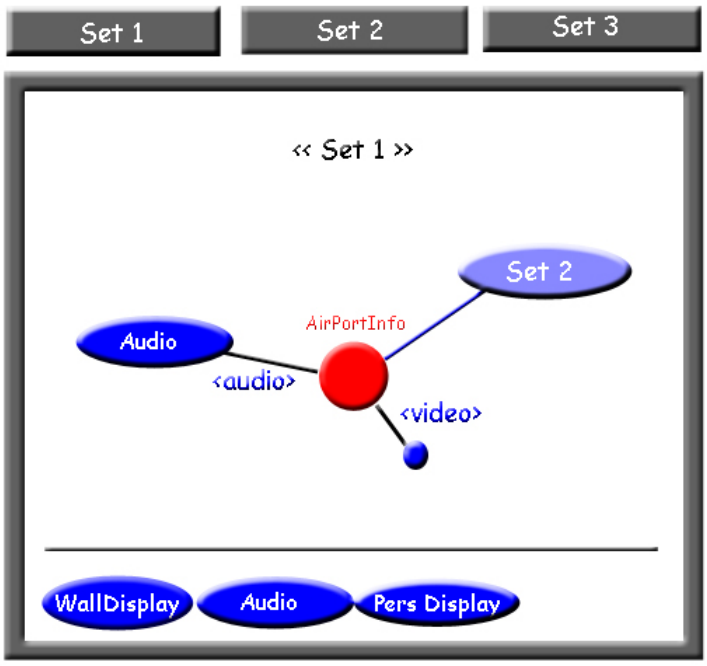


Figure B.2: GUI iteration 1b

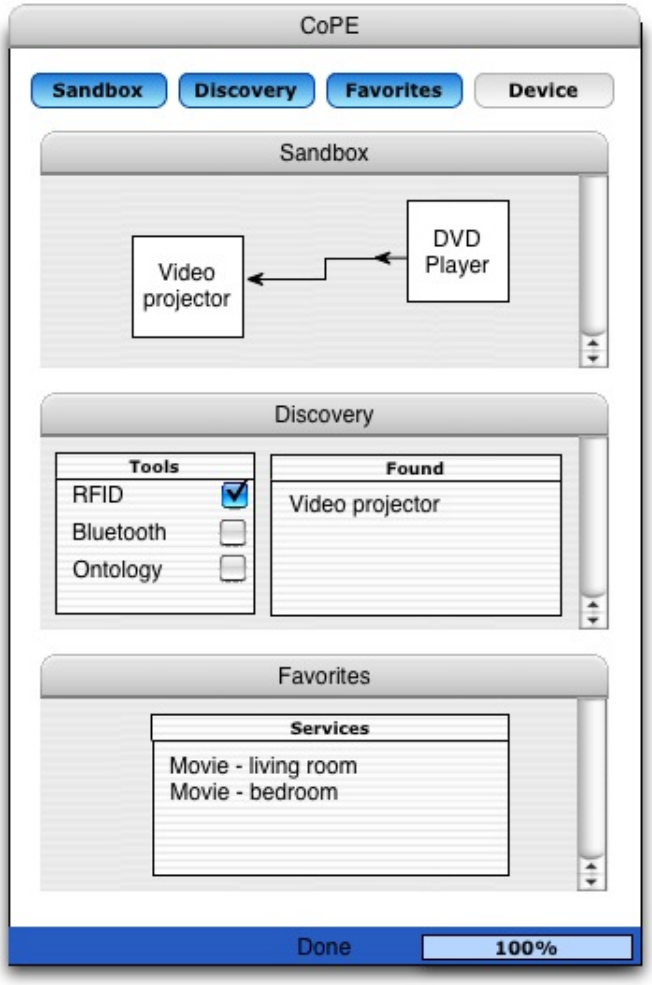


Figure B.3: GUI iteration 2a

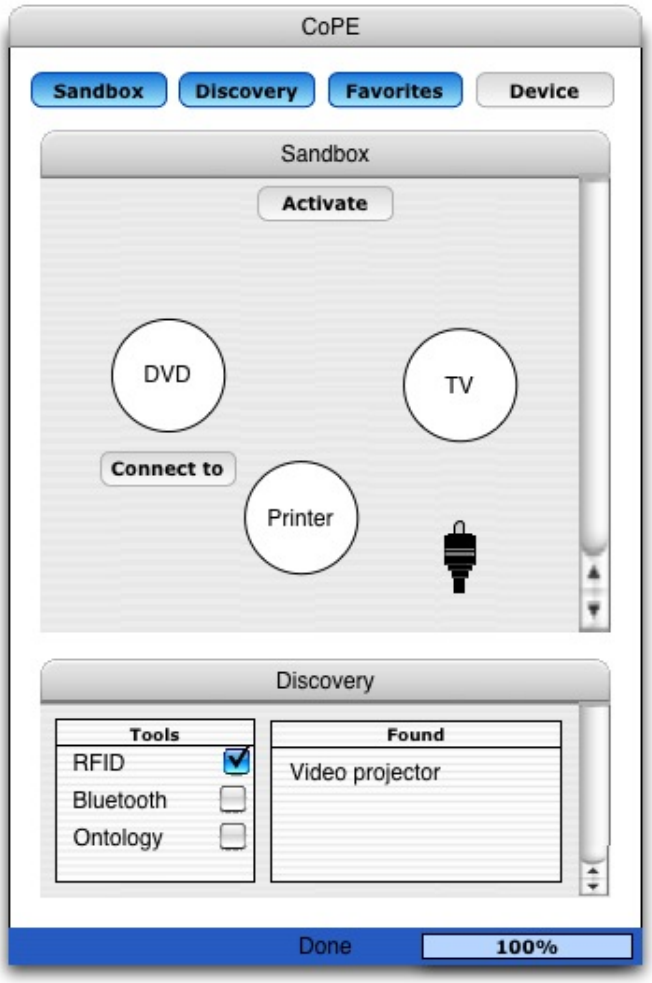


Figure B.4: GUI iteration 2b

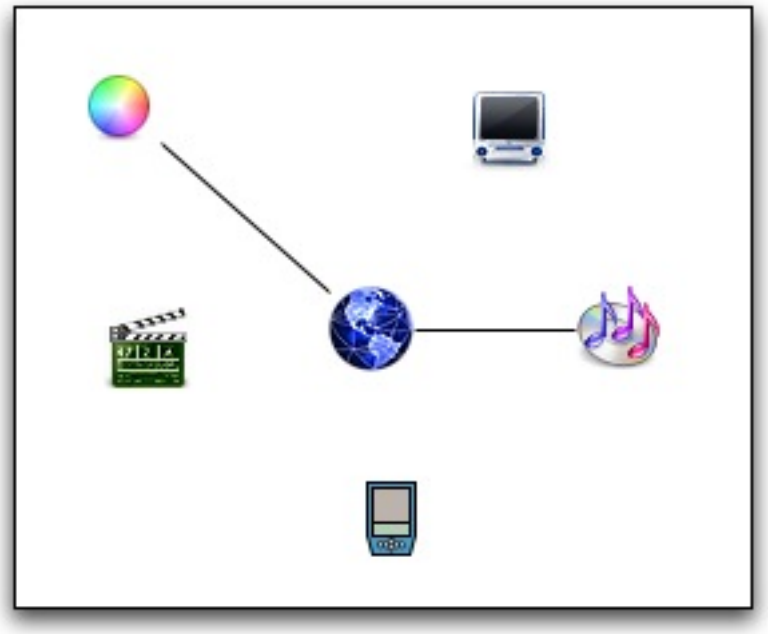


Figure B.5: GUI iteration 3

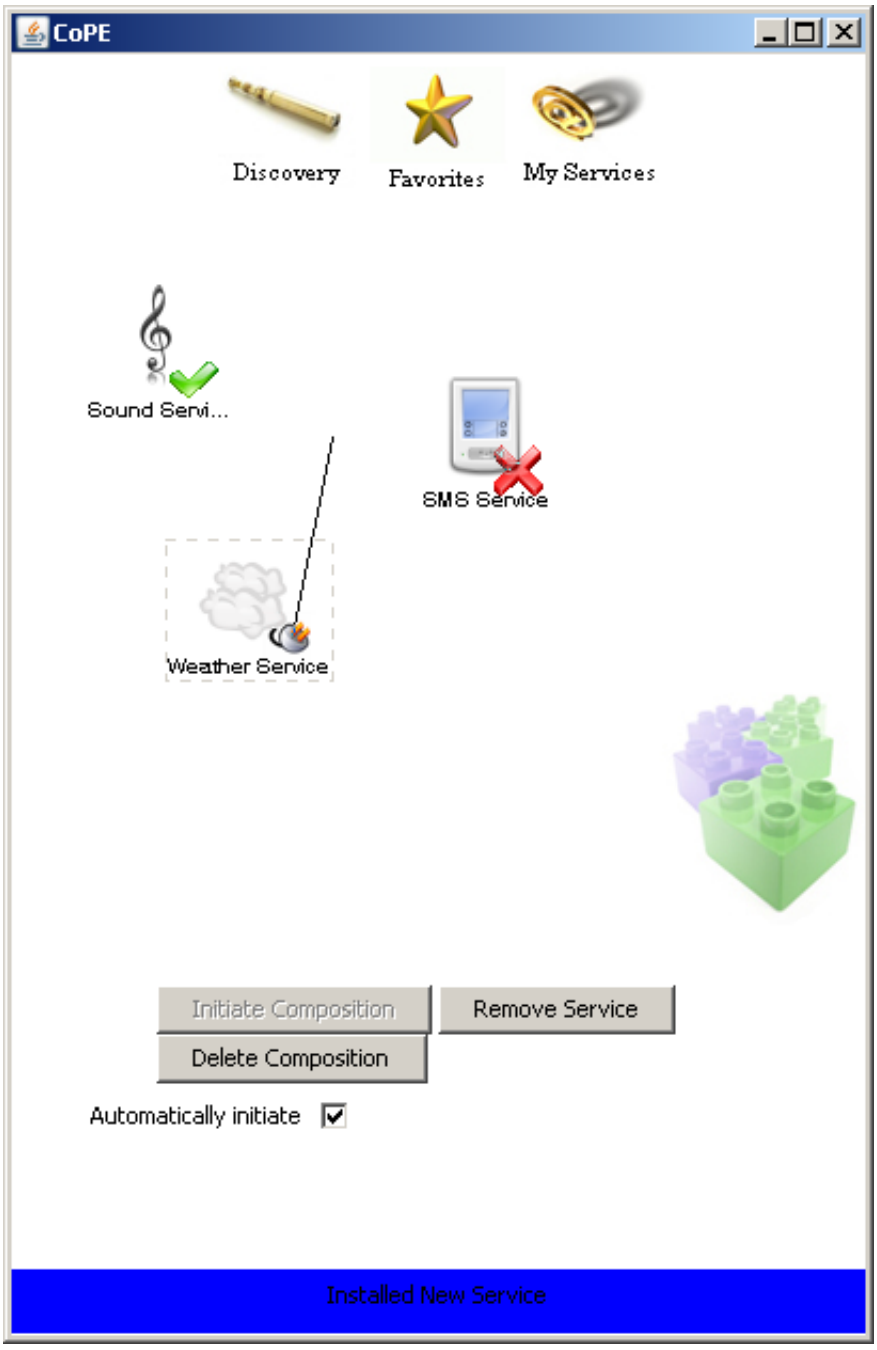


Figure B.6: GUI iteration 4a

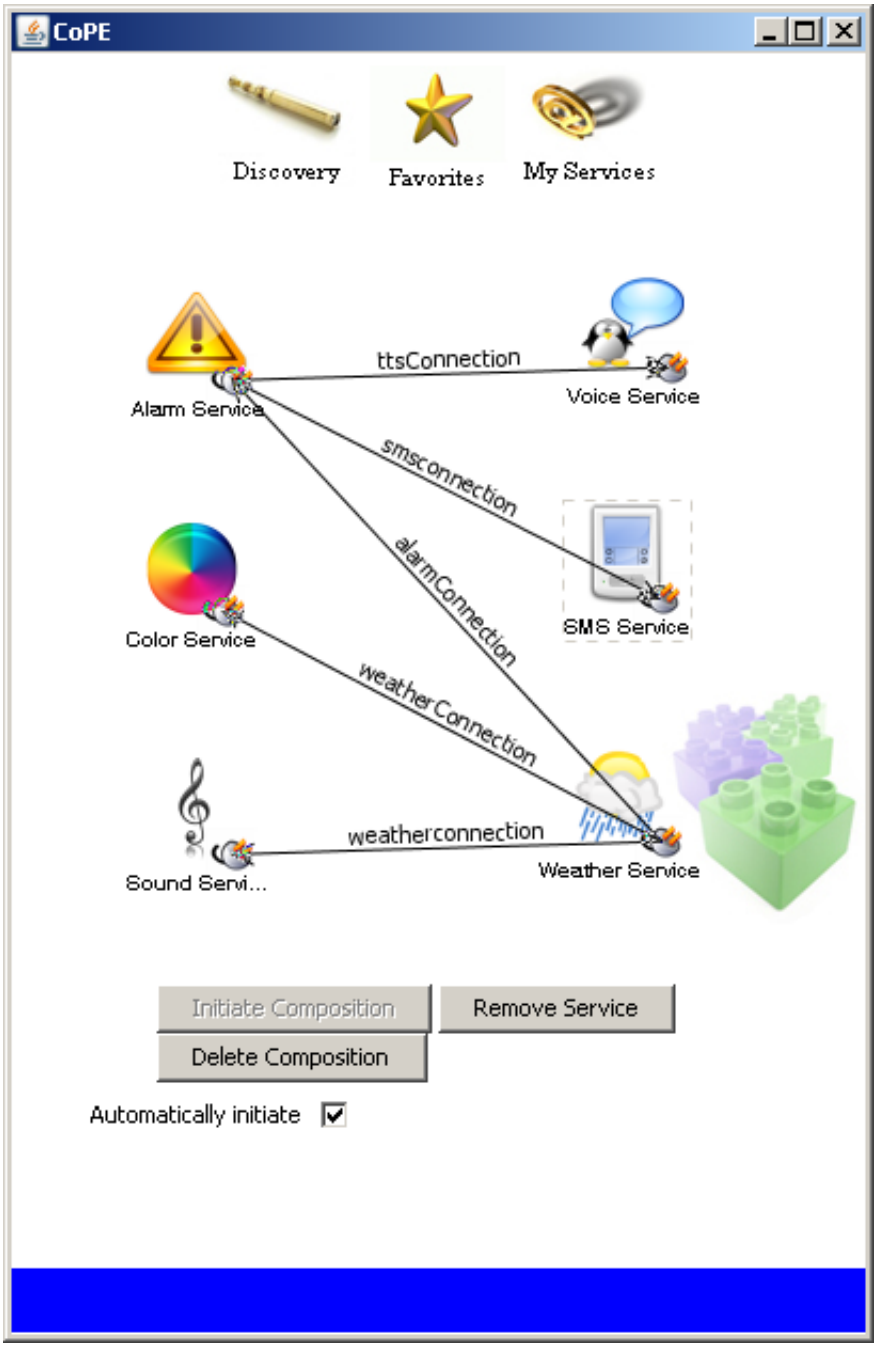


Figure B.7: GUI iteration 4b

Bibliography

Gregory D. Abowd and Elizabeth D. Mynatt. Charting past, present, and future research in ubiquitous computing. *ACM Trans. Comput.-Hum. Interact.*, 7(1):29–58, 2000. ISSN 1073-0516. doi: http://doi.acm.org/10.1145/344949.344988.

Gregory D. Abowd, Christopher G. Atkeson, Jason Hong, Sue Long, Rob Kooper, and Mike Pinkerton. Cyberguide: a mobile context-aware tour guide. *Wirel. Netw.*, 3(5):421–433, 1997. ISSN 1022-0038. doi: http://dx.doi.org/10.1023/A:1019194325861.

Apache. Apache license 2.0. URL `http://www.apache.org/licenses/LICENSE-2.0`.

A. Crabtree, T. Rodden, T. Hemmings, and S. Benford. Finding a place for ubicomp in the home. *Proceedings of the 5th International Conference on Ubiquitous Computing*, pages 208–226, 2003.

N. Dahlback, A. Jonsson, and L. Ahrenberg. Wizard of oz-studies – why and how. 1993. URL `citeseer.ist.psu.edu/45570.html`.

Scott Davidoff, Min Kyung Lee, Charles Yiu, John Zimmerman, and Anind K. Dey. Principles of smart home control. *UbiComp 2006*, pages 19–34, 2006.

Anind K. Dey, Raffay Hamid, Chris Beckmann, Ian Li, and Daniel Hsu. a cappella: programming by demonstration of context-aware applications. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 33–40, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-702-8. doi: http://doi.acm.org/10.1145/985692.985697.

K. Gajos. Rascal-a resource manager for multi agent systems in smart spaces. *Proceedings of the Second International Workshop of Central and*

Eastern Europe on Multi-Agent Systems (CEEMAS 2001), pages 111–120, 2001.

Krzysztof Gajos, Harold Fox, and Howard Shrobe. End user empowerment in human centered pervasive computing. *Proceedings of Pervasive*, 2002.

S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura, and E. Jansen. The gator tech smart house: A programmable pervasive space. *Computer*, 38(3):50–60, 2005.

Lars Erik Holmquist, Friedmann Mattern, Bernt Schiele, Petteri Alahuhta, Michael Beigl, and Hans-W Gellersen. Smart-its friends: A technique for users to easily establish connections between smart artefacts. In *Ubicomp 2001: Ubiquitous Computing: Third International Conference Atlanta, Georgia, USA, September 30 - October 2, 2001, Proceedings*, page 116. Springer Berlin and Heidelberg, September-October 2001.

http://tuim.inf.puc rio.br/interact2007/home.php. Interact 2007, conference. URL `http://tuim.inf.puc-rio.br/interact2007/home.php`.

http://www.thinlet.com/. Thinlet. URL `http://www.thinlet.com/`.

J. Humble, A. Crabtree, T. Hemmings, K.P. Åkesson, B. Koleva, T. Rodden, and P. Hansson. Playing with the bits-user-configuration of ubiquitous domestic environments. *Proceedings of the Fifth Annual Conference on Ubiquitous Computing, UbiComp2003, Seattle, Washington, USA*, pages 12–15, 2003.

Min Kyung Lee, Scott Davidoff, John Zimmerman, and Anind Dey. Smart homes, families, and control. *Proceedings of Design and Emotion*, 2006.

Kim Halskov Madsen. A guide to metaphorical design. *Commun. ACM*, 37(12):57–62, 1994. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/198366.198381.

Adam Marx. Using metaphor effectively in user interface design. In *CHI '94: Conference companion on Human factors in computing systems*, pages 379–380, New York, NY, USA, 1994. ACM Press. ISBN 0-89791-651-4. doi: http://doi.acm.org/10.1145/259963.260514.

Mark W. Newman, Jana Z. Sedivy, Christine M. Neuwirth, W. Keith Edwards, Jason I. Hong, Shahram Izadi, Karen Marcelo, Trevor F. Smith, Jana Sedivy, and Mark Newman. Designing for serendipity: supporting end-user configuration of ubiquitous computing environments. In *DIS*

'02: Proceedings of the conference on Designing interactive systems, pages 147–156, New York, NY, USA, 2002. ACM Press. ISBN 1-58113-515-7. doi: http://doi.acm.org/10.1145/778712.778736.

Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The context toolkit: aiding the development of context-enabled applications. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 434–441, New York, NY, USA, 1999. ACM Press. ISBN 0-201-48559-1. doi: http://doi.acm.org/10.1145/302979.303126.

M. Satyanarayanan. Pervasive computing: vision and challenges. *Personal Communications, IEEE [see also IEEE Wireless Communications]*, 8(4): 10–17, 2001.

Dag Svanaes and William Verplank. In search of metaphors for tangible user intefaces. In *DARE '00: Proceedings of DARE 2000 on Designing augmented reality environments*, pages 121–129, New York, NY, USA, 2000. ACM Press. doi: http://doi.acm.org/10.1145/354666.354679.

X. Wang, J.S. Dong, C.Y. Chin, S.R. Hettiarachchi, and D. Zhang. Semantic space: An infrastructure for smart spaces. *IEEE Pervasive Computing*, 3 (3):32–39, 2004.

Mark Weiser. Some computer science issues in ubiquitous computing. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):12, 1999. ISSN 1559-1662. doi: http://doi.acm.org/10.1145/329124.329127.