NTNU
Norwegian University of
Science and Technology

Dynamic Management of Software Components in a Ubiquitous Collaborative Environment

Yngvar Kristiansen

Master of Science in Computer Science
Submission date: July 2010
Supervisor: Babak Farshchian, IDI

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

Problem Description

Traditionally, computer systems have managed files stored on hard drives. In ubiquitous computing computers have to manage a distributed collection of files, services, devices, objects etc. stored in different locations and accessed by many users. In UbiCollab, the collection of services, devices, objects, files etc. that the user owns or uses are represented as the user's Service Domain. A user's service domain will typically contain his/her devices at home or office (e. g. stereo, TV, lights, computers). UbiCollab Service Domain Management subsystem allows users and the system to manage what is in each user's service domain, how to add or remove things for the service domain, etc. The objective of this project task is to further develop UbiCollab Service Domain Management subsystem.

The main research questions to be answered by this project task:

- How can we extend existing service management architectures to support user-centered and community-based service management?
- What technologies and architectures are most suitable for implementing user-centered and community-based service management?
- How can we evaluate the usability and utility of user-centered and community-based service management? What are the most compelling scenarios?

Expected deliverables:
- Scenarios for user-centered service management.
- Architecture and design for UbiCollab Service Domain Management subsystem.
- Extensions, in form of Java code, to the existing Service Domain Management subsystem in UbiCollab.
- Extensions, in form of Java interfaces, to the existing Service Domain Management APIs in UbiCollab.
- Implementation (in Java) and testing (in JUnit) of user-centered Service Domain Management subsystem in UbiCollab.
- GUI (In Java) for allowing users to control UbiCollab Service Domain Management.

Assignment given: 22. February 2010
Supervisor: Babak Farshchian, IDI

Abstract

The key **motivation** of this thesis is to find innovative solutions for facilitating the deployment of ubiquitous systems, with the purpose of making technology supported collaboration an easier task. Users, being in a ubiquitous environment, continuously encounter new resources that might provide some value. As the number of these resources increase, the management of them will be a central task in a ubiquitous computing system.

The **problems and challenges** discussed in this thesis are related to continuous and unpredictable changes in the ubiquitous environment, which makes it difficult for users to retrieve appropriate software for utilizing resources. We also discuss the challenge of managing resources, and sharing them between users.

The **research questions** in this thesis are:
RQ-1: *How can we extend existing service management architectures to support user-centered and community-based service management?*
RQ-2: *What technologies, architectures and platforms are the most suitable for implementing user-centered and community-based service management?*
RQ-3: *How can we evaluate the usability and utility of user-centered and community-based service management? What are the most compelling scenarios?*

The **contributions** in this thesis are, correspondingly:
C1: We have made a solution proposal and an implementation of an improved service management system, which is based on earlier works of the Ubicollab platform.
C2: Four items were found suitable: 1. The deployment model used by distribution platforms for mobile applications (such as AppStore and Android Market), 2. OSGi, 3. R-OSGi, and 4. HTTP-based communication using Java Servlets.
C3: The evaluation of such systems can be done using a three-step process that includes: 1. Examining the system's fulfillment its requirement specification. 2. Compare the system's functionality with that of a scenario-described ideal system. 3. Create applications that demonstrate the utility of the system.

Preface

This thesis represents the final work for the degree of Master in Computer Science at the Norwegian University of Science and Technology (NTNU), Department of Computer and Information Science (IDI). The work was done during the period of Mars to July in 2010.

This work is a contribution to the Ubicollab platform, which aims at supporting technology-based collaboration between people. This work suggests and implements a system for user-centered and community-based management of resources in an ubiquitous environment.

I would like to thank my supervisor, Babak Farshchian, at IDI, NTNU for very helpful guidance and feedback during my work. I also would like to thank the other students working on Ubicollab, who provided helpful introduction to the project, as well as enlightening discussions.

Trondheim, July 19, 2010

Yngvar Kristiansen

Table of contents

List of figures

List of tables

Abbreviations

OS	Operation system
UI	User Interface
GUI	Graphical User Interface
CSCW	Computer Supported Cooperative Work
AmI	Ambient Intelligence
SOA	Service Oriented Architecture
OSGi	Former Open Service Gateway initiative, but now the abbreviation doesn't stand for anything.
SDM	Service Domain Manager
CI	Collaboration Instance
API	Application Programming Interface

Chapter 1 Introduction

The sections in this chapter are structured the following way:

- 1.1 presents the motivation for this thesis.
- 1.2 describes the relation between this thesis and other research.
- 1.3 presents the research questions to be answered by this thesis.
- 1.4 explains how this thesis fulfills the design-science research method.
- 1.5 outlines the structure of the rest of this thesis.

1.1 Motivation

The key motivation for this thesis is to find innovative solutions for facilitating the deployment of ubiquitous computing systems, and by that, make technology supported collaboration an easier task. Users, being in a ubiquitous environment, continuously encounter new resources that can enhance the experience of the users. These resources might enhance user interaction mechanisms not found on a mobile device, or might provide additional computational resources. These kinds of ambient resources are the cornerstone of ubiquitous computing as envisioned by Weiser [1]. As the users encounter an increasing number of these resources in their daily life in a ubiquitous computing environment, the management of these resources becomes a central task for the ubiquitous computing system.

A more fundamental motivation behind this is thesis, is a hope for increased focus on interoperability and ubiquitous collaboration. As more and more developers, managers and customers see the benefits of ubiquitous collaboration, more and more ubiquitous applications will hopefully emerge, which in turn will bring us nearer Weiser's vision of the disappearing technology.

1.2 Research context

This project is part of the Ubicollab project. The Ubicollab project unifies the research fields Computer Supported Cooperative Work (CSCW) and Ambient Intelligence (AmI). This has resulted in the development of the Ubicollab platform, whose aim is to use technology to aid ubiquitous collaboration among users. The unifying concept for CSCW and AmI in Ubicollab is the *Human Grid*, which "denotes a collection of (geographically distributed) users and the resources each of them has available in their physical vicinity" [2]. The Ubicollab platform supports discovery of resources (for instance a projector or a smart coffee machine), service management, context awareness and user profiles, amongst other things.

The main basis for this thesis is two earlier works in the Ubicollab project, namely the thesis of Johansen [3] and the thesis of Mora [4]. Related to this thesis, the main contribution of Johansen's work was designing and implementing a user-centered service discovery and management system, i.e. a foundation for the Ubicollab platform. The main contribution of Mora's work, related to this thesis, was making an implementation of Ubicollab for mobile devices, and doing development of new and existing Ubicollab applications and plug-ins.

This thesis continues the work on the service management system in Ubicollab.

1.3 Research questions

The research questions for our work are the following:

RQ-1: *How can we extend existing service management architectures to support user-centered and community-based service management?*

RQ-2: *What technologies, architectures and platforms are the most suitable for implementing user-centered and community-based service management?*

RQ-3: *How can we evaluate the usability and utility of user-centered and community-based service management? What are the most compelling scenarios?*

1.4 Research method

The research method used in this thesis follows the design-science paradigm [5]. In this paradigm, new knowledge is gained by creating and evaluating an innovative artifact (for instance a prototype of a system).

Hevner et al. [5] argue that there are two complementary but distinct paradigms in information systems' research, with design-science paradigm being one of them. The other paradigm is the behavioral-science paradigm, which "*seeks to develop and verify theories that explain or predict human or organizational behavior*". It is in other words a more theoretical approach to expanding knowledge than the design-science paradigm.

Hevner et al. suggest seven guidelines for conducting effective design science research. However, it is emphasized that the guidelines should be used to *assist* researchers, and not be used as a strict scheme.

Guideline	Description
Guideline 1: Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
Guideline 2: Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
Guideline 5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Guideline 6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Table 1.1: Design-Science Research Guidelines

The following sections describes how each of the guidelines in Table 1.1 is followed in this thesis.

Guideline 1: Design an artifact

The artifact of this thesis is the Service Domain Manager (SDM), described in Chapter 5.

Guideline 2: Problem relevance

The problem of this thesis and its relevance is described in section 1.1, “Motivation”, and section 1.2, “Research context”.

Guideline 3: Design evaluation

The evaluation method for the SDM is described in section 6.1, "Methods used for evaluation".

Guideline 4: Research Contributions

Chapter 3 presents related research to this thesis, and that there exists no earlier work similar to the main research contribution of this thesis, namely the Service Domain Manager.

Guideline 5: Research Rigor

The construction and evaluation method of the SDM are described in section 5.2 and 6.1, respectively.

Guideline 6: Design as a Search Process

This thesis represents one iteration in the work of the Ubicollab project. Section 1.2 describes the context of where this iteration fits.

Guideline 7: Communication of Research

This report presents the research done, and thus serves as the "communication" of the research. Section 6.5 describes in detail the research contributions of this work.

1.5 Thesis structure

The reminder of this thesis is structured as described in the following paragraphs:

Chapter 2: Problem analysis

Chapter 3: State of the art

Chapter 4: Solution proposal

Chapter 5: Implementation

Chapter 6: Evaluation

Chapter 7: Conclusion and further work

Chapter 2 Problem analysis

This chapter is structured the following way:

- Section 2.1 presents a scenario in order to identify problems related to management of resources in a ubiquitous, collaborative environment.
- Section 2.2 and 2.3 describes the general problems and challenges that can be extracted from the scenario.
- Section 2.4 presents requirements for a solution to the discussed problems.

In ubiquitous computing, there is a vision that a user should be able to freely move from place to place while interacting with resources in a computing environment, not paying attention to tools or configuration [1]. Ideally, the user should be able to interact with resources themselves, not via surrogate software-based interfaces representing these resources. This is, in other words, a biologic approach, where a user do not carry any technical device, and is identified using for instance iris [6] or fingerprint recognition [7]. There are, however, many technical as well as ethical challenges that must be met before this vision can become true.

This thesis does not use the biological approach to ubiquitous computing, but does instead take the approach of using a *mobile device* as a personal gateway for connecting the user to the ubiquitous environment. The mobile device will thus represent the identity of the user, instead of the user's fingerprint or iris as in the biological approach. As a result, users must carry their mobile device in order to interact with and be identified by resources in the ubiquitous environment.

The problems discussed in this chapter lie within the research field of CSCW (Computer Supported Cooperative Work) and ubiquitous computing.

For the rest of this thesis, we will assume that users access resources through their default mobile device, and that resources can be accessed through an API.

2.1 Problem scenario

This thesis will use one scenario in order to identify problems related to management of resources in a ubiquitous, collaborative environment. We will refer to this scenario as the *ProblemScenario* throughout this thesis.

Eve is a university employee, and is today going to the university library for a meeting with Joe and Ashley. She has booked a meeting room, where they will meet at 3pm and discuss some budgets.

After arriving the library, Eve wants to print one of the budgets , so she uses her smart phone to search for printers on the wireless network. She finds several printers, but then realizes she does not have any way of opening and printing the budget, which is a spreadsheet in the XLS file format. Additionally, her smart phone needs a device driver for installing the printer.

She decides to use the smart phone's online "marketplace", where she can search for applications to download, and hopefully find one that can open the spreadsheet. She issues a search for "print", and gets an overwhelming amount of hits. Clueless about which application to pick, she chooses the first on the list and installs it. She runs it, but unfortunately it only supports printing of images*. She tries another application, and this time she gets one that only works with Bluetooth-enabled HP-printers. She decides to read through the applications' descriptions more carefully, and finally finds an application "XLSManager", that claims that it "prints all XLS spreadsheets!". She runs the application and checks that it is able to find the printers in the library. She also noticed the application "PDF Printer", that is able to convert spreadsheets to the PDF file format. She installs that application too, in case she needs it during the meeting.*

Eve still needs the device driver for accessing the printer on her smart phone. Not wanting to browse a lot of applications like she just did, she asks the library's receptionist for help. The receptionist gives her a URL to a webpage that contains the device drivers for the printer, so Eve is able to install the printer. After installation, she is has to enter a username and password for the printer, as only employees and students are allowed access, so she enters the username and password for her employee account.

Eve suddenly notices that her estimated remaining battery time is alarmingly low - only two hours left. The battery was fully charged this morning, and she suspects that one or more of those new applications she installed might be the cause. She decides to uninstall the two printing applications that couldn't print her budget. Having had her smart phone for a year, she has to navigate through a lot of other applications before she finally is able to find those two applications and uninstall them. Afterwards, the battery still only got two hours left. Eve wants to keep XLSManager and PDF Printer, so instead of uninstalling them, she attempts to check if they can be configured to use less battery in some way. Both applications have placed their configuration in different places in the GUI, but Eve finally manages to find three checkboxes in XLSManager: "Always search for new printers, "Search using the wireless network", and "Search using Bluetooth". XLSManager has already found the library's printers, so she turns off all of the checkboxes, and is delighted to see that the phone's estimated battery life now is at around 11 hours.

Eve decides to try to print out the budget. The printer that her smart phone was able to find, was named "Library printer Room F032", and Eve asks a receptionist for directions to the printer. Eve finally prints the budget, and sees a notice above the printer: "Price per page: $0.20. Please pay at reception desk." Eve heads back to the reception and pays.

At 3pm, Joe and Ashley arrives, and Eve gets back to the meeting room, a couple of minutes late. At the end of the meeting, they have agreed to use Joe's version of the budget. Joe wants to print some copies of the budget for archiving. Because Eve has access to the library printer, she offers him to print his budget, and tell him to send his budget on e-mail. When receiving the e-mail, Eve sees that Joe unfortunately uses the XLSX format, a more recent version of the XLS format, and the XLSManager isn't capable of reading this format. Tired after a long day, Eve doesn't quite want to do the whole search process again for finding a working application, so she kindly asks if Joe can print his budget at a later time.

2.1.1 Summary of problems

Table 2.1 summarizes the problems in the ProblemScenario.

Scenario problem summary
- Eve wanted to open an XLS-file and print it. - She made an application search for "print", and found an overwhelming amount. - She had to make three attempts before finding a suitable application.
- Eve attempted to uninstall two recently installed applications, but used some time to navigate through a great number of previously installed applications before she could find and uninstall the two applications.
- Eve needed a device driver for the printer, and asked the receptionist for help, in order to avoid spending a lot of time searching for the driver.
- Eve wanted to view the configuration for two other applications, but she used some time to find the configuration because each of the GUIs of the applications looked different, and Eve was unfamiliar with both.
- Eve tried to find the library printer, but had to ask the receptionist for directions.
- Joe needed to print his budgets on the library printer, but could not print because he did not have username and password to the printer.

Table 2.1: Scenario problem summary.

2.2 Continuous and unpredictable changes in the ubiquitous environment

This section describes challenges for a ubiquitous, collaborative computing platform in which the user's context changes continuously and unpredictably. For a user moving from place to place, new resources are found while other disappear. At the same time, the platform must provide a user-friendly interface to enable interaction between the user and the environment and its resources. This challenge needs to be concretized further, and based on the ProblemScenario, two new groups of challenges have been identified:

- Challenges related to retrieving the appropriate software when the user's mobile device discovers new resources, described in section 2.2.1.
- Challenges related to managing software already installed on the user's mobile device, described in section 2.2.2.

2.2.1 Retrieving appropriate software for utilizing a resource

In the ProblemScenario, Eve two times needed to find the appropriate software for utilizing the library printer. One time she needed an application that could open and print her spreadsheet, and the other time her smart phone needed a

device driver in order to communicate with the printer and be able to issue the print job.

The ProblemScenario scenario identifies a general need in ubiquitous computing: As the user's context change and new resources are discovered, the user will need a way to interact with these new resources. This can be done by installing a resource controller[1] and one or more applications that utilize the resource controller's capabilities. Eve did this by installing a printer driver (resource controller) and the XLSManager (application). The relation between these concepts is illustrated in REF _Ref266661068 \h Figure 2.1.

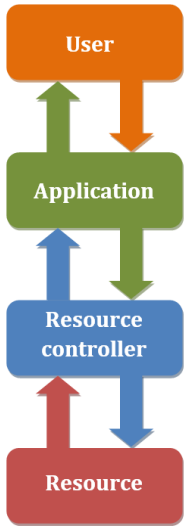


Figure 2.1: The relation between resources, resource controllers applications and users.

However, resources are made by different manufacturers and have their own communicate protocols. As a result, they are accessed through different APIs. Consequently, a separate resource controller for each different API has to be made. For instance, in a series of printers from the same manufacturer, it is common that each printer comes with separate printer driver.

In order for users to interact with new resources, an appropriate resource controller is needed for allowing applications to access the resource, and one or more suitable applications are needed for utilizing the resource controller's capabilities. The challenge is how the user's mobile device can retrieve the

[1] See the glossary.

appropriate resource controller and applications in a user-friendly way. Using the ProblemScenario as an example, Eve needed a simple way of retrieving the printer driver and the XLSManager.

2.2.1.1 Possible approaches and their problems

This section describes possible approaches to how a user can retrieve resource controllers and applications that enable a user to utilize a resource. The approaches are based on functionality available on smartphones today ([8],[9]). Then problems with these approaches are described.

The user can find and download software by using one or more of the following approaches:

1. Searching the Internet using a standard search engine like Google, Yahoo, or similar.
2. Searching by using an Internet site dedicated for searching for software, like download.com.
3. Searching by using a dedicated search application (like App Store[2] or Android Market).

The problems with these approaches are described in the following sections.

Long search time

Searching for software may or may not take a lot of time. This depends on several factors.

First, determining good search terms can reduce the search time significantly. In the ProblemScenario, Eve searched for the single word "print", while searching for "print xls" probably would have given more relevant and fewer[3] hits. The result was that she spent extra time on browsing and installing irrelevant software. This factor is especially true for point 1, because the search database is much larger for a general search engine than a search engine specialized for software.

[2] Not to be confused with the Apple Store, which is an online store, while App Store is a application on iPhone and iTouch for searching and downloading applications for the device.

[3] Assumed that the search algorithm replaced spaces with the boolean AND operator.

Second, a typical search engine is usually unable to tell with certainty that what the user is looking for does not exist. In the ProblemScenario, if there existed no applications that could print spreadsheets , Eve may have spent a lot more time searching, browsing and installing unsuitable applications without finding anything useful.

Third, if the name of the software does not reveal what it does, it may be prioritized lower by the search engines and thus be harder to find.

Which application to choose?

A search may give multiple results, illustrated in Figure 2.2. If that is the case, how does the user choose between the results? In the ProblemScenario, Eve's search for "print" gave too many search results, which made her install two unwanted applications. The user maybe does not care about, or know how to interpret, the subtle differences of the search results, and just wants some software that works. In that case, a list with too many results represent unnecessary complexity for the user.

Figure 2.2: Multiple choices adds unnecessary complexity for the user.

2.2.2 Managing installed software

So far problems related to acquiring software for interacting with resources have been discussed. The following sections will describe challenges related to the management and administration of this kind of software after it has been deployed on the user's mobile device.

2.2.2.1 Controlling system resource consumption

In the ProblemScenario, the battery consumption were significantly increased because the application XLSManager was constantly using the wireless network and Bluetooth. In software systems where software components can be installed, the need arises for managing these components. Examples of managerial tasks can be activating, deactivating, uninstalling and configuring a component. Without such possibilities, there is a risk that components use the mobile device's restrained hardware resources unnecessarily. Some negative effects of this can be faster draining of the device's battery, and a slower user interface.

With management capabilities, the user (or a software agent) can disable or remove unnecessary software components, and configure and optimize existing components – in other words maintaining the system so it uses the mobile device's resources effectively.

2.2.2.2 A usability challenge with resource management

As the user gets more and more resource controllers and applications, she will increasingly need some way to manage them, as mentioned in the last section (2.2.2.1). If she is to do managerial actions on more than a few of these components, the mobile device must offer a quick and user-friendly way for doing this.

For instance: In the ProblemScenario, Eve was able to uninstall the two applications she did not need. What if she had 40 applications and resource controllers, and wanted to uninstall all resource controllers used the week, or all applications that had been installed while she was at home? Smartphones today do normally provide only basic sorting capabilities for organizing and managing installed applications.

2.3 Sharing of resources

This section describes challenges related to enabling users to share resources with other users.

The ProblemScenario described how Joe was not able to print his budget on the library printer, because he did not have a username and password to the printer. However, how would the scenario look like if Eve was able to *share* the

library printer with Joe? Would such an approach be faster than the option Eve found too time-consuming?

The option Eve discarded was printing Joe's budget by finding a new application that supported printing files in the XLSX file-format Joe used. Eve decided this process possibly would take more time than she was willing to spend, because of the problems she had with finding a working application (described in 2.2.1.1). Further, Joe was not able to print the budget himself, as the printer was password protected, and Eve probably did not want to reveal her employee account's username and password.

The problem in the scenario would have been solved if there was some way Eve could *share* the library printer. If she could use her smart phone as some kind of proxy server, acting as a communication link between the printer and Joe's mobile device, it should be possible for Joe to utilize the printer without needing a username and password.

This thesis will attempt to find out if this kind of sharing is possible, and if so, how it can to be solved in a user-friendly way.

Given that it is possible to share resources as described above, new challenges and needs arise:

- **How can a the host of a shared resource monitor control the usage of the resource?** For instance, say that Eve shares the printer with five users. If one of them prints a large amount of expensive color pages, Eve may want to find out who did the print to make sure that person pays for the print. She may also want to limit the maximum number of pages that any of the client users can print.
- **Is it possible to share not only resources, but also services that runs on a user's mobile device?** If sharing of resources are possible, it may be possible to share of services as well. Sharing of services could possibly enable users to share functionality on their mobile devices with other users. For instance, if Joe was able to share a service that could convert XLSX files to XLS files, Eve could have utilized this shared service instead of finding and downloading a separate application that could open XLSX files.

2.4 Requirements specification

Based on the problems and challenges identified and discussed in this chapter, it is possible to deduce requirements for a solution that can cope with these problems and challenges. These requirements are presented in Table 2.2. They are high-level, and not meant to completely specify a system, but detailed enough for a system that solve the problems described in this chapter.

Each requirement in Table 2.2 has:

- An ID and an abbreviation, which is used for referral later in this thesis.
- A complexity degree, which is either L(ow), M(edium) or H(igh).
- A priority, with the same possible values as for the complexity.

ID, Abbreviation	Description	Complexity	Priority
R-1, Mobiliity	**The system must be possible to operate and be fully functional on a mobile device.**	M	H
R-2, Appropriate-SoftwareRetrieval	**When the user's mobile device discovers a resource, the system must be able to retrieve appropriate software that can utilize that resource.** By appropriate software, we mean a software component that is able to interact with the resource.	H	H
R-2.1, ListOfRetrievable Software	**The software components found by the system to be appropriate for the discovered resource, shall be listed to the user, so that the user can select which component(s) to retrieve.**	M	H
R-2.1.1, ContextAware-List	**The system must be able to utilize attributes from the user's context in order to produce a list of appropriate software components.** For instance, the user must be able to list software that is used in the physical vicinity.	H	H

ID, Abbreviation	Description	Complexity	Priority
R-2.1.2, Prediction-SortedList	**The system shall sort the list of software components based on predictions of the user intents and needs.**	H	M
R-2.2, NoSoftware-Exists	**If no appropriate software for a resource exists, the system must be able to clarify this to the user.**	M	M
R-3, ControlRes-Consumption	**The system must offer some means to control consumption of the system's hardware resources.** Examples of hardware resources are CPU, memory, storage and bandwidth.	M	M
R-4, ListSoftware-Organized	**The system must be able to show installed software components in an organized way.** Organizing components can be done for instance by sorting the components based on properties like location, time of installation, usage frequency, etc.	M	H
R-5, ShareSoftware-Components	**The user must be able to share the abilities of software components that supports sharing to other users** For instance, User A can share a converter service or a printer so that User B can use it as well. This requirement does not require that all software components must be shareable.	H	H

ID, Abbreviation	Description	Complexity	Priority
R-5.1, UseComponent-OnBehalf	**A user using shared software components (a client user), must be able to utilize the shared component on behalf of the host user.** For example: If User A shares a printer resource, and User B sends a print task to the printer, the printer sees the origin of the print task as User A.	H	M
R-5.2, MonitorShared-Usage	**A device that shares a software component, must be able to monitor the usage of the component.** Usage originating from the device that is sharing the software component does not need to be monitored. Only usage originating from other devices must be able to be monitored.	M	M
R-5.3, ControlShared-Usage	**The host user of a shared software component must be able to control usage of the resource.** For instance for limiting the number of users using the resource at once, or for limiting the time a software component should be shared.	H	M

Table 2.2: Requirements for a solution.

Chapter 3 State of the art

This chapter discusses the state of the art related to the problems identified in Chapter 2. Each research item presented in this chapter contributes to solving one or more problems from Chapter 2.

The structure of this chapter is as follows:

- Section 3.1: Presents Ubicollab, the platform we will implement a solution on.
- Section 3.2: Describes research for the problems in section .
- Section 3.3: Describes research for the problems in section .
- Section 3.4: Summarizes how the research solves the problems from Chapter 2.

3.1 Ubicollab

This section describes how Ubicollab contributes to solving the problems of Chapter 2. A more thorough and foundational description of the Ubicollab project is presented in the Ubicollab whitepaper [2]. For the rest of this thesis, we will use terms from the whitepaper, whereof the most relevant are found in the glossary of this thesis.

3.1.1 Background

Ubicollab is a platform for supporting technology-based collaboration between people. The Ubicollab project has its root in the research fields of Computer Supported Cooperative Work (CSCW) and ubiquitous computing. The Ubicollab project's "*research agenda for ubiquitous collaboration is set to take advantage of research results in CSCW and AmI in order to create better solutions for supporting natural collaboration in co-located or distributed groups*" [2].

3.1.2 Platform

Ubicollab is based on a service oriented architecture (SOA). It is not dependent on a central server, so all services reside on the user's UbiNode (mobile device). As a consequence, communication between UbiNodes are done in a peer-to-

peer fashion. Ubicollab's platform implementation consists of independent *Platform Services*.

See Figure C.1 (page 106) for an overview of the architecture of the Ubicollab platform.

3.1.3 Communication with resources

A UbiNode can communicate with resources in the user's physical vicinity or on the Internet. Because resources are not implemented using a standard API, the UbiNode's communication with resources is done by using *Proxy Services*. Proxy Services are in other words similar to resource controllers, see section 2.2.1.

Section 2.2.1 discussed users' difficulties of retrieving appropriate for utilizing a resource. The current implementation of Ubicollab supports resource discovery [4], but is not able to retrieve appropriate Proxy Service(s) when discovering a resource. After a resource has been discovered, no behavior is implemented for dealing with the new resource. Unless a Proxy Service for that resource has previously been installed, the UbiNode (and the user) will not be able to utilize the resource, because of the lack of a standard API. UPnP [10] and DNLA [11] are approaches that address the lack of a standard API, but before these solutions become widely deployed, Proxy Services are used for communicating with resources.

3.1.4 Platform implementation: OSGi

Ubicollab is implemented using OSGi [12], which is a platform for building and running dynamic and modular applications in Java. OSGi runs on top of the Java Virtual Machine, and provides a runtime environment for applications. Applications are implemented using one or more modules called *bundles*. Bundles are packaged as JAR files. Bundles support dynamic deployment in that they can be installed, started, stopped, updated and uninstalled during runtime, that is, without stopping the host application or the Java Virtual Machine.

Resources are accessed via Proxy Services, and Proxy Services are implemented as bundles. Because the dynamic capabilities of bundles, resources can be added and removed from the user's UbiNode as the user moves around and discovers new resources. This is a crucial capability in ubiquitous computing,

and is the reason for why OSGi has been selected as platform for Ubicollab in earlier and this work.

3.2 Continuous and unpredictable changes in the ubiquitous environment

3.2.1 Retrieving appropriate software for utilizing a resource

In short, section 2.2.1 described problems that can occur when a user wishes to acquire new software on her mobile device. Several approaches for software retrieval do exist, and are discussed in this section.

3.2.1.1 A recommender application for mobile devices

Woerndl et al. [13] implements a client program for mobile devices, from now on called *AppRecommender*[4], that makes software application recommendations to the user, based on the user's context. The user interface (Figure 3.1) makes the user select which kind of recommendation she wants, e.g. "nearby services" or "interesting services". The user interface then shows a list of applications based on the user's choice. For instance, AppRecommender can suggest a restaurant-guide application based on the user's location.

The motivation for creating AppRecommender, was the wish to make it easier for users to get information that is relevant amongst the huge amount of information that exists today.

Relevance to problem analysis

How does AppRecommender relate to the problems discussed in section 2.1? Solution 3 in section 2.2.1.1 suggests that a relevant application can be fetched using a dedicated search application, and AppRecommender is such a search application.

AppRecommender can address the problem of long search time by utilizing the user's context to recommend applications to the user, so that the user does not have to spend a lot of time finding suitable search terms. How useful the recommendations are, is dependent on the recommendation algorithm. A useful recommendation algorithm will select a few applications that all matches

[4] Woerndl et al. does not name their implementation, and is added here for increased readability.

the user's needs, while a poor algorithm will output a long list of applications that have no value for the user.

Figure 3.1: The mobile client program lists recommended applications.

3.2.1.2 Automated installation of devices in a ubiquitous healthcare system

Jung et al. [14] describe a system for automatic installing of newly discovered bio-sensor devices used in a ubiquitous healthcare system.

Part of the goal of the work of Jung et al. is to utilize a bio-sensor on a mobile device. In order to do that, the mobile device needs a device driver to be able to communicate with the resource. This problem is similar to the problem described in section 2.2.1, "Retrieving appropriate software for utilizing a resource".

Jung et al. solve this problem by having letting the mobile device recognize the bio-sensor, request it's device driver from a device driver server, and install the driver. This process is automatic, so no user intervention is needed.

Jung et al. offers an intriguing approach for quickly finding the appropriate software based on a resource, by using a mapping between resources and their appropriate software. However, this work assumes exactly one matching device

driver for each resource, while in our problem, there may exist multiple appropriate software components for a resource.

Figure 3.2: Bio-sensor devices.

3.2.1.3 Mobile barcode readers

Microsoft Tag [15] and NeoReader [16] are two applications for mobile devices, that let the user take pictures of visual symbols, *barcodes*, in order to access mobile web content. Each barcode is unique, making it possible to convert it to a unique URL. For instance, a user can take a picture of a barcode in an article about Norway in a travel magazine, and the user's web browser then opens a web page allowing the user to see videos of Norway and order tickets.

Mora's implementation of Ubicollab includes a similar function like Microsoft Tag and NeoReader. It allows a user to take a picture of a barcode in order to identify a resource [4].

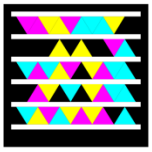


Figure 3.3: A barcode from Microsoft (a Microsoft Tag).

Relevance to problem analysis

The barcode scanning functionality enables a user to take a picture of a barcode attached to resources in her physical surroundings, and retrieve the appropriate software based on the unique barcode. This can be done by linking the barcode to a URL to a web page that either downloads a software component supporting the resource, or by listing different software so the user can make a choice.

In other words, this approach contributes to how the appropriate software can be fetched for utilized a resource.

The problem of long search time is addressed by this approach, because the barcode is able to quickly identify a resource. Because a barcode is unique, this method also enables the mobile device to either automatically install the appropriate software for the identified resource, or present several suitable software choices to the user.

3.2.1.4 Application distribution systems for mobile phones

Major mobile operation systems usually provide an application distribution system, in the form of an application, along with the operation system. Several distribution systems for mobile devices exist, perhaps the most known are Apple's App Store and Google's Android Market, if measuring by the number of available applications available [17]. These systems are shown in Figure 3.4 and Figure 3.5, respectively. We'll investigate the most central features of the distribution systems here, including features from all major application distribution systems.

There are some common functions that all or most of the distribution systems offer. If the user wants a new application, she can start the distribution application (App Store, Android Market, etc.) and browse applications. Applications can be sorted, usually by category (sports, games, etc.), popularity, date and paid versus free. The user can also search by using search terms. Nokia Ovi Store suggests applications to the user via a "Suggested for me"-feature, which makes recommendations based on what kind of application the user has earlier downloaded.

These functions do address the problem of getting a suitable application. Sorting applications by category, popularity and similar do help narrowing down the amount of applications the user has to investigate in order to find the right application. However, the vast number of applications existing in these distribution systems can still make this difficult. In the biggest distribution market, the AppStore, there were as of June 2010 about 220 000 applications, and about 5 000 applications are added every week [18].

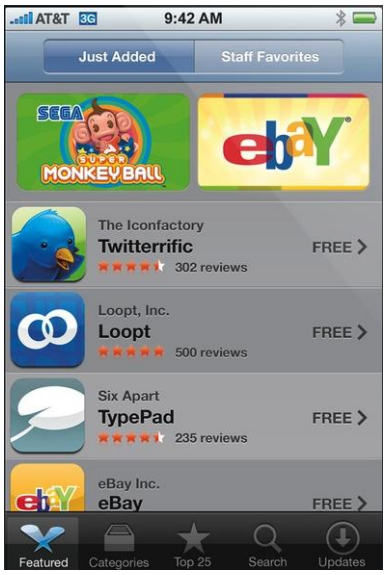


Figure 3.4: App Store.

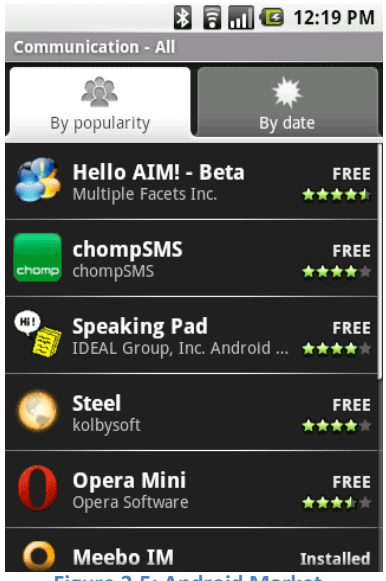


Figure 3.5: Android Market.

3.2.1.5 Application distribution systems for desktop PCs

Utilizing applications is one of the main purposes for a user using an operation system (OS). A user may want to fetch another application, and some OSes provide functionality for doing this.

Windows

In Windows (version Vista and later), there is no direct way of finding and retrieving new applications. However, if a user tries to open a file with a file extension that is not associated with an application, Windows suggests to search for a suitable application (Figure 3.6). Choosing this option brings the user to a web page which lists links to applications that can handle the file (Figure 3.7). Clicking on any of these links redirects the user to the application producer's web page.

This solution enables the user to find a suitable application. The retrieval process is a quick process, only two mouse clicks are needed to get to a web page that can offer an application. It is then up to this web page how easy it will be for the user to fetch the application. For instance, is the user able to download the application at once, or must she navigate through several pages in order to download the application?

Windows' solution is quick and user-friendly. However, for our purpose, the solution is probably too basic, as the application selection mechanism is based only on a file extension.

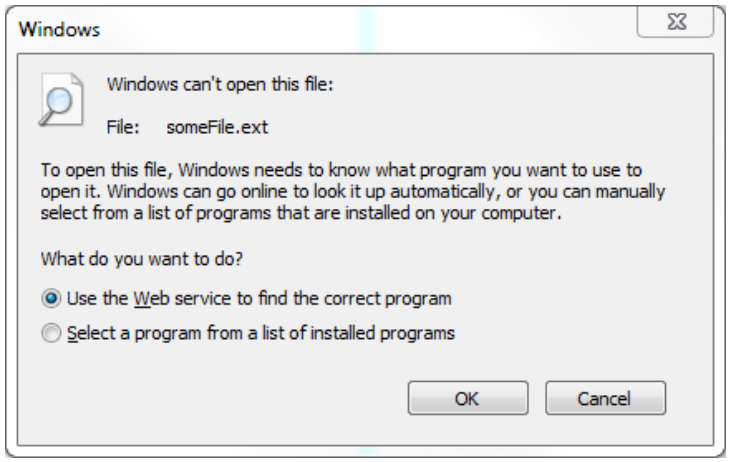


Figure 3.6: Windows suggestions when opening an unknown file type.

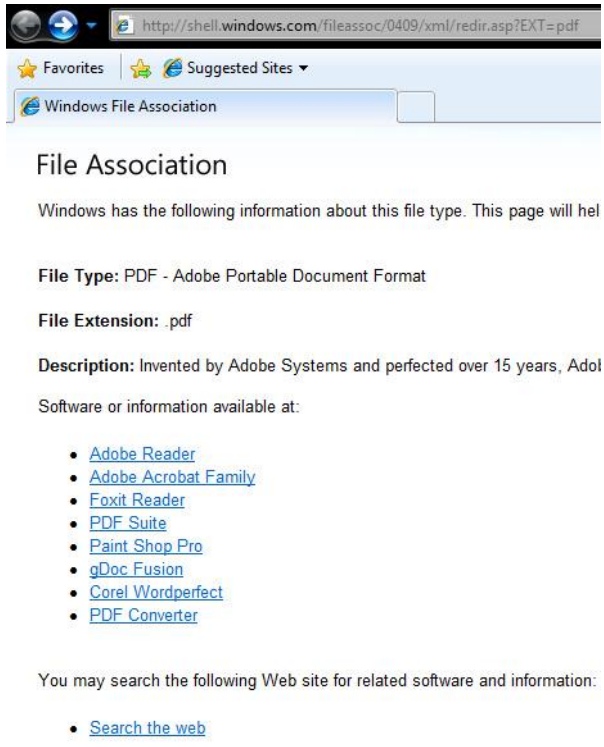


Figure 3.7: Application suggestions for handling a file type.

Linux/Ubuntu

Ubuntu is a user friendly [19] Linux distribution for desktop PCs. It includes the "Ubuntu Software Center" (Figure 3.8), which is an application management portal that lets the user in few steps search (within a category or not) for applications, get their description, download and install them. The user can also browse applications in different categories without entering any search terms. After download and installation, the application is put in the correct category of the Ubuntu drop down panel (equivalent to the Windows' start menu). The Ubuntu Software Center also provides an overview of applications already installed, and gives the option to uninstall existing applications.

The Ubuntu Software Center (Figure 3.8) gives the user a consistent and user-friendly way to install, get an overview of, and uninstall applications. However, it does not recommend or rank applications in any way, nor does it utilize the user's context, so user must investigate categories, application names and application descriptions in order to find an application that suits her needs. The categories help reduce search time, but there may still be many alternatives within a category.

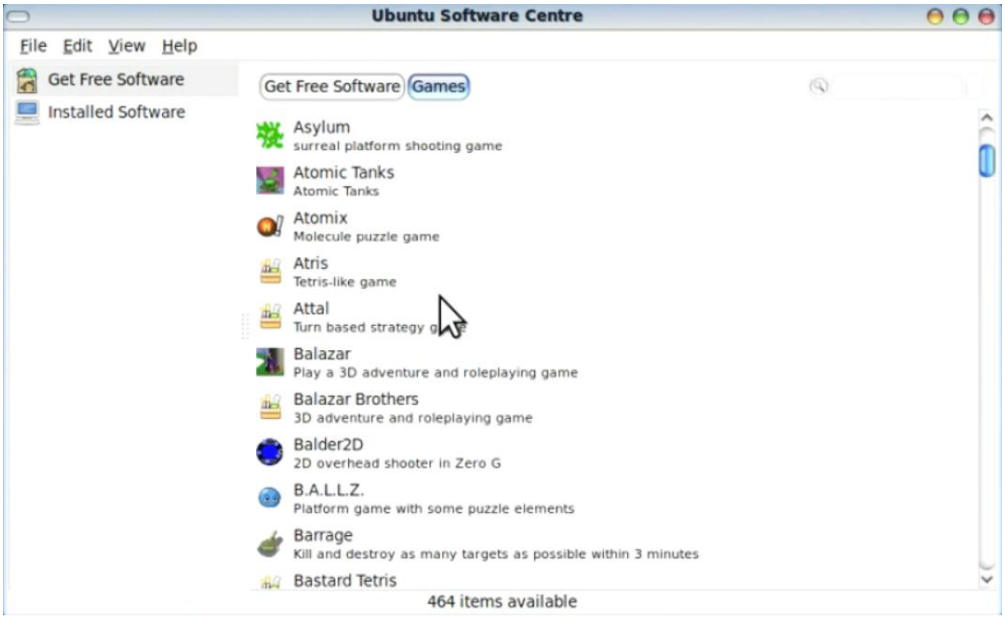


Figure 3.8: Browsing games in Ubuntu Software Center.

3.2.1.6 OSGi bundle repositories

A bundle repository is a web resource that contains pointers to bundles that clients can download. The web resource can be a XML file that contains a database with metadata about bundles, or a human-readable web site. The intention of a bundle repository is to provide a trusted site that clients can use for discovering bundles [20].

Bundle repositories represent a good idea for how software can be retrieved on a user's mobile device. When a user discover a resource, metadata about the resource can be provided to the bundle repository, which will respond with one or more suitable bundles that the user can download and install on her mobile device.

3.2.2 Managing existing software

In short, the following issues related to software management were discussed in section 2.2.2:

- Possibility for increased consumption of a device's hardware resources as more and more software components are installed.
- How can the user interface enable the user to effectively manage many software components?

Section 3.2 discussed solutions related to acquiring software, and this section continues by examining how management of existing software can be done.

3.2.2.1 Resource consumption

This section will examine how resource consumption can be controlled by managing software on the Android platform and the Windows (version 7) platform. The reason for choosing these two platforms is that they include working, recent solutions for software management and resource consumption control.

Google Android

When installing software (in form of applications) on Android, there will be increased usage of the mobile device's resource because applications use disk storage, memory, CPU and bandwidth. This results in faster draining of the mobile device's battery.

In addition to applications, the Android OS enable the use of services, which also may drain the device's battery.

The user is able to control resource consumption in several ways. The user can force an application to stop, clear its cache, and uninstall it. As for services, the user can start, stop and configure them. The service itself is responsible for what configuration is available, and it is thus possible for a service to include settings for resource consumption. For instance, a weather forecast service can include a setting for updating every hour instead of every 30 minutes.

The functionality for resource control mentioned so far is included by the Android OS. However, there are a great number of applications available on the Android Market that offer additional functionality for resource control. One examples is application Advanced Task Manager [21], which offers, amongst other things, automatic killing (shutting down) applications or services when they launch, and stopping multiple selected applications or services at once.

Windows

Most versions of the Windows OS can get slower as more and more applications are installed. By slower, we mean increase in the response time of the user interface, as well as the startup time of the OS and for applications. Some reasons for the increased slowness includes (but is not limited to):

- After installation of an application, it can add itself to Window's list of applications that are to be run on startup. The same applies to services. This increases usage of the hardware resources.
- As additional hardware devices are installed on the PC that runs Windows, more and more device drivers will need to be continuously run. This increases the CPU and disk usage.

There are many other reasons why Windows can get slower. However, the listed reasons above are included here because they describe things that can happen in most systems that can run applications and services.

Windows includes software management functionality that aids the user in controlling resource usage. The concepts of this management functionality can possibly also be implemented in a solution to the problems in Chapter 2. We

will in the following paragraphs examine the management functionality that Windows offers.

Uninstall software: A simple solution to removing unnecessary software, is uninstalling it. Through the control panel, Windows lets the user uninstall applications.

Remove startup applications: Through the tool "msconfig.exe", the user is able to select which applications can be included on startup of Windows.

Reduce an application's visual settings: Editing the compatibility settings of an executable file, a user can set that the application should run in low-color mode, low-resolution mode, and some other selections for reducing the visual quality.

Block network access: Windows Firewall allows the user to completely block network access for an application, which will eliminate any bandwidth usage.

Process management: Through Task Manager, a user can:

- Kill (quit) processes that is currently running.
- Set priority of a process, which sets how much CPU-time a process gets.
- Set affinity of a process, which sets the number of CPU cores available to a process.

Service management: The service management tool (started with executable services.msc) includes the following functionality:

- Starting, stopping, restarting, pausing, and resuming of services.
- Setting the startup type, which can be one of the following:
    - Automatic (delayed): The service is started first when Windows has completed its startup process, in order to speed up the startup.
    - Automatic: The service is started at system startup.
    - Manual: The service is started when needed (lazy startup).
    - Disabled: The service is prevented from starting.

Hardware management: The Device Manager in Windows allows the user to manage hardware attached to the PC Windows is running on. Several functions that can be applied on a hardware device are available for the user:

- Uninstall: Removes the device driver for communicating with the device from the OS.
- Disable: Removes the visibility of the hardware device from the OS so it is not recognized or identified by any applications. After disabling, enabling the device again is possible.
- Update device driver: Updates the device driver software (the user can search for or specify a location to a newer device driver).
- Configure: Configuring a device driver lets the user control internals of the hardware. The device driver itself provides what settings the user can change.
- Power management: For some hardware, the user can set if the OS should be able to turn off the hardware device in order to save power.

OSGi bundle management

Section 3.1.4, "Platform implementation: OSGi", mentioned that bundles can be installed, started, stopped, updated and uninstalled during runtime. These actions are repeated here because they can be used for controlling resource consumption. For instance:

- If a bundle uses too much CPU-time, it can be stopped.
- If the user's mobile device at a point has too many bundles installed, they can be uninstalled.

The capability to do these actions at runtime without having to reboot other systems is another attractive feature for controlling resource consumption.

3.2.2.2 Usability

Section “A usability challenge with resource management” (within section 2.2.2), described a scenario in which the problem was to manage many software components. None of the solutions examined so far provide a solution or ideas for solutions to this problem, because they do not utilize the user’s context in order to manage existing software. Matsumoto et al. [22] and

Kamisaka et al. [23], however, present solutions that can be used for further work in this thesis, and will be examined here.

Matsumoto et al. implemented a context-aware user interface, ordering applications in a menu based on location and time. The menu contained 15 applications that the user could start. An experiment showed that the method was able to predict the user's preferred application better than using only usage frequency for prediction. The approach of Matsumoto et al. does not fully solve our problem (in section 2.2.2), because too few context attributes are used (physical location and time), and our problem will need additional attributes like existence and properties of nearby resources and logical locations (Collaboration Spaces in Ubicollab). However, the solution of Matsumoto et al. is an indicator that a context-aware user interface is actually an improvement to non-context-aware user interfaces.

Kamisaka et al. try to solve the problem with the difficulties users can have with trying to find an appropriate application on a mobile device, in complex menus and with many sub-levels. They attempt "*to realize a context-aware UI which enables a user to access an appropriate application including an infrequently-used one (calls, e-mails, camera, navigation system, applications installed by a user, and all other capabilities the mobile phone is equipped with) quickly according to his/her situation.*"

Their problem is in some way similar to our current problem, which is how to let the user manage many software components. When there are many software components to choose from, the user would benefit from a smart user interface that is able to predict which software component she is looking for.

They concluded that time (which hour of the day) and the last action executed were useful attributes when making a ranked list of possible actions for the user. However, useful attributes to use in Ubicollab are likely to be different because of numerous reasons. They analyzed long-term usage data of users when using mobile phones for normal usage in order to find attributes such as time and last action executed. Users most likely did not spend a significant amount of time on managing software applications in a ubiquitous context, and thus it is reasonable to assume that the usage data would be different if analyzed similarly on the Ubicollab platform.

What we can learn from the work of both Matsumoto et al. and Kamisaka et al. is that:

- It is probably wise to include attributes from the user's context when creating a user interface that gives the user numerous options.
- Logs of the user's actions (navigation, execution, etc) can be used to choose the order of elements in the user interface.

3.3 Sharing of resources

This section discusses research that can be useful when creating a solution to the problem of sharing a user's context and capabilities, described in section 2.3.

3.3.1 MobiShare

MobiShare is "*a middleware system that supports publishing, advertising and semantic discovery of mobile resources encapsulated in services*" [24]. The system is accessed through access points (*CAS*es in Figure 3.9) which communicates with their own administration server. Each access point operates within a physical range (called a *cell*). Users access this system through a user interface on their mobile device.

Users are able to discover services through a context-aware search process. A user can specify a search (like "book taxi"), which sends a search request to a nearby access point. The access point maintains an internal list of available services, and when a user issues a search, the access points does a context-based search on this service list. The results are shown to the user as a list of services. Context parameters supported include location and orientation (e.g. north).

Users are not limited only to consuming services, they can define their own services as well. The client user interface includes the *service definition tool*, which enables users to share their own services, such as "*a cab driver is accepting reservation on his mobile phone or a sports fan sharing pictures on his PDA*" [24]. In order to formally define what the service can do, the user must specify input and output parameters, and which part of a static and previously defined taxonomy it belongs to. The taxonomy is used for instance to place a service in the category "taxis", which is a sub-category of "travel".

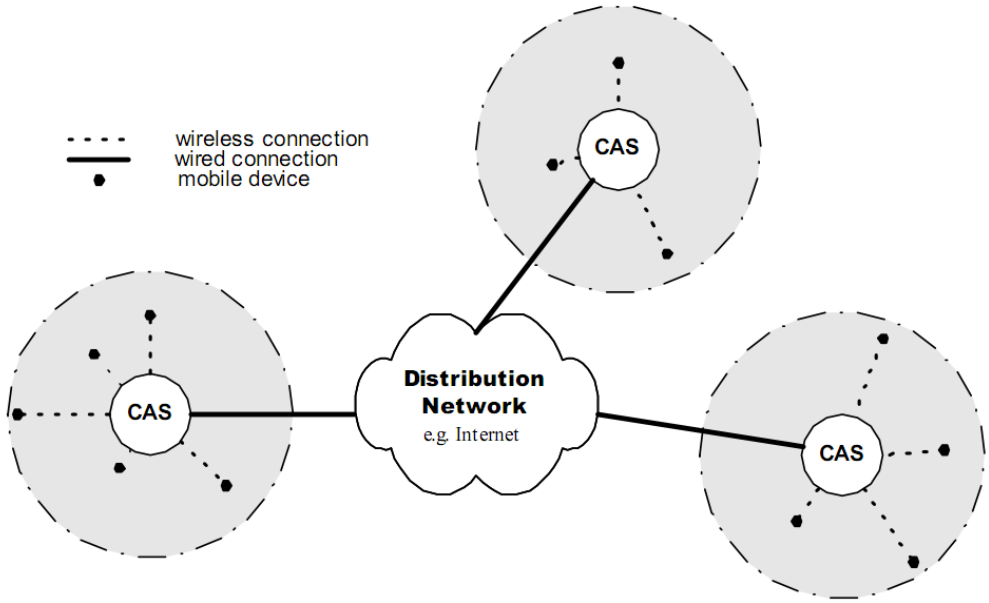


Figure 3.9: The network topology in MobiShare.

The ProblemScenario described a scenario where collaboration could have been simplified if it was possible for Eve to share the library printer with Joe. MobiShare does provide this through the service definition tool described above.

MobiShare is an interesting contribution to our problems. However, the MobiShare architecture uses central access points in order to provide service management (discovery, searching, publishing and removal of services). If these are not available, the system goes down. No peer-to-peer approach is used, which is required for our solution.

3.3.2 R-OSGi

R-OSGi enables transparent, remote access to OSGi services [25]. A remote OSGi service can be retrieved via the network from another peer, using the OSGi framework as if it was a local service. When the host of the service registers the service to the OSGi framework, the host has to specify a single property to indicate that it should be able for remote access. R-OSGi is deployed as a normal OSGi bundle.

In effect, R-OSGi enables creation and usage of distributed applications. And more importantly, R-OSGi enables software components to be shared among users, because software components can be deployed as bundles.

The performance of R-OSGi is comparable or better than that of RMI or uPnP, which are similar distribution mechanisms [25]. The bundle has been tested on a number of OSGi implementations and hardware platforms.

R-OSGi was suggested as future work for the Ubicollab platform by Mora [4].

3.4 Summary

See Table 3.1 for a summary of the research found relevant for the problems in Chapter 2. Empty cells means that the system does not support the requirement. A cell's text content explains how the system solves the requirement. For instance, we can see that the BioSensor-system supports requirement R-2.

System / Req-ID	Ubicollab	OSGi	AppRecommender	BioSensor	Barcode readers	Mobile OSes	Desktop OSes	OSGi Bundle Repositories	Matsumato, Kamisaka, et al.	MobiShare	R-OSGi
R-1, Mobiliity	Yes	Yes									
R-2, Appropriate-SoftwareRetrieval			Yes	Yes	Yes. Possible by creating a web page that shows suitable software for the identified resource.		Yes. Windows finds programs based on file extensions.	Yes. Can output bundles based on resource properties.			
R-2.1, ListOfRetrievable-Software			Yes		Yes. Same as above (R-2).	Partly. The OSes can list software for retrieval, but not based on a resource.	Yes. Windows views a web page with a list of suitable software.	Yes. Possible to browse repositories on web.			
R-2.1.1, ContextAware-List			Yes		Partly. A list can be made based on the time, and what the tag can tell about the context.				Yes	Yes	

System / Req-ID	Ubicollab	OSGi	AppRecommender	BioSensor	Barcode readers	Mobile OSes	Desktop OSes	OSGi Bundle Repositories	Matsumato, Kamisaka, et al.	MobiShare	R-OSGi
R-2.1.2, PredictionSortedList			Partly. Possible by imple-menting a prediction algortihm.		Partly. Possible to some extent by producing web content based on the tag ID and time of scan.				Yes	Yes	
R-2.2, NoSoftware-Exists			Yes				Windows notifies the user when the file extension is not known.		Assumed yes.	Yes	
R-3, Control-ResConsumption		Yes				Yes	Yes		.		
R-4, ListSoftware-Organized						Yes	Yes	Yes	Yes	Unknown	
R-5, ShareSoftware-Components										Yes	Yes
R-5.1, UseComponent-OnBehalf										Unknown	Yes

System / Req-ID	Ubicollab	OSGi	AppRecommender	BioSensor	Barcode readers	Mobile OSes	Desktop OSes	OSGi Bundle Repositories	Matsumato, Kamisaka, et al.	MobiShare	R-OSGi
R-5.2, MonitorShared-Usage										Assumed yes	Yes
R-5.3, ControlShared-Usage										Unknown	Yes

Table 3.1: How the state-of-the-art contributes to a our requirements for a solution.

Chapter 4 Solution proposal

This chapter describes a Ubicollab Platform Service that shall solve the problems described in Chapter 2, “Problem analysis”.

The contents is structured in the following way:

- Section 4.1 presents a scenario that illustrates a solution to the earlier problems, as well as an analysis of the scenario to see how it solves these problems.
- Section 4.2 and 4.3 state the suggested functionality for the new system. The sections are structured similarly to their respective sections in Chapter 2 and Chapter 3.

4.1 Solution scenario

4.1.1 Scenario

This section presents a scenario that starts the same way as the ProblemScenario, and shows how the system presented in this chapter solves the problems presented in Chapter 2. This scenario is referred to as the *SolutionScenario* for the rest of this thesis. Within the scenario, there are references, written **in bold**, to GUI sketches for a solution.

Eve is a university employee, and is to have a budget meeting with Joe and Ashley at 3pm in a specific meeting room at the university library. They are to discuss some budgets.

*At the library, Eve wants to print her budget. She finds her UbiNode, and makes a search for printers on the wireless network. She finds one called "Library printer Room F032", selects "Find services for this resource", selects "Device driver (1)" , and is able to get a Proxy Service that her UbiNode needs for accessing the printer (***Figure 4.1***). After installation, she is prompted for a username and password for the printer, as only employees and students are*

allowed access, so she enters the username and password for her employee account.

*Eve notices that she does not have any Proxy Service for opening and printing the budget, so using her UbiNode, she starts Service Domain Manager and selects "see installed services". She then chooses to list services for newly installed devices, which that are tagged with "spreadsheet". She installs the first on the list, "DocManager" (***Figure 4.2***). She also thinks they might need converting the budgets to PDF during the meeting, so she makes Service Domain Manager to retrieve a list of Proxy Services tagged with "spreadsheet" and "PDF". She selects the first application suggested, named "XLS-to-PDF", which tells in its description that it can convert XLS files to PDF.*

*Eve suddenly notices that the estimated remaining battery time on her UbiNode is only two hours, even if it was fully charged this morning. She suspects one of the new Proxy Services to cause the problems, so she uses Service Domain Manager to get a list of recently installed Proxy Services (***Figure 4.3***). Eve chooses XLS-To-PDF, and finds the configuration button in the menu as she is used to, and clicks it. However, she does not find any setting in the configuration which would make XLS-To-PDF significantly drain the battery. She repeats the same process for DocManager, and is able to spot three check boxes, "Always search for new printers, "Search using the wireless network", and "Search using Bluetooth". The UbiNode has already found and installed the library's printer, so Eve turns off all the checkboxes. Afterwards, she is glad to see that the estimated remaining battery time has increased to 11 hours.*

Eve checks the description of the Proxy Service "Library printer Room F032", and is able to find directions to Room F032, where the printer is located. While walking, Eve issues her printout and receives a message that says her credit card has been charged by a total of $1.40, $0.20 per page. At Room F032, Eve fetches the papers from the printer tray, and goes back to the meeting room, in good time before Joe and Ashley arrives.

At the end of the meeting, they have agreed upon using Joe's version of the budget. Joe wants to print some copies of the budget to hand over to the client at a later meeting, so Eve creates a Collaboration Instance named "Group Meeting". She then shares the library printer with the Group Meeting *Collaboration Instance, so Joe's UbiNode is able to find and install it (***Figure**

4.4*) . Joe prints his budgets via Eve's UbiNode, and Eve is notified that her credit card has been charged by a total of $1.20, $0.20 per page.*

4.1.2 Analysis of solved problems

In Table 4.1, the problems in the column "ProblemScenario" are taken from Table 2.1, "Scenario problem summary.".

Problem	ProblemScenario	SolutionScenario
Section 2.2.1: Users discover new resources when they move around in a ubiquitous environment. In order to utilize new resources, new software is needed. **How can users easily aquire appropriate software?**	- Eve needed a device driver for the printer, and **asked the receptionist for help, in order to avoid spending a lot of time searching for the driver.**	**See Figure 4.1.** The UbiNode automatically found a device driver.
	- Eve wanted to open an XLS-file and print it. - She made an application search for "print", **and found an overwhelming amount**. **- She had to make three attempts before finding a suitable application.**	**See** Figure 4.2**.** - Eve was **able to use the installed printer and a word (type) in order to find a suitable application**.
Section 2.2.2: After installing software components, new issues arise. These issues are related to **increased resource consumption** of the user's mobile device and **user-friendliness when managing many software components**.	- Eve attempted to uninstall two recently installed applications, but **used some time to navigate through a great number of previously installed applications** before she could find and uninstall the two applications..	**See** Figure 4.3**.** - Eve **quickly found the two recently installed Proxy Services** in the SDM by choosing the "recently installed" option.
	- Eve wanted to view the configuration for two other applications, but she **used some time to find the configuration** because each of the GUIs of the applications looked different, and Eve was unfamiliar with both..	**See** Figure 4.3**.** - Eve were able to **quickly check the configuration** of the two recently installed Proxy Services, because she found the configuration button for both of them at the same general place in the menu.

	- Eve tried to find the library printer, but **had to ask the receptionist for directions**.	**See** Figure 4.3**.** - Eve **opened the description** of the Proxy Service for the printer and found directions to the printer.
Section 2.3: When users collaborate, there may be a need for **sharing resources or functionality** between their mobile devices (for instance a nearby projector, or some file converter functionality). Users utilizing a shared resource, may also need to **use the resource on behalf of the user sharing it**.	- Joe needed to print his budgets on the library printer, but **could not print because he did not have username and password to the printer**.	**See** Figure 4.4**.** - **Eve shared the library printer with Joe**. - Joe found the shared printer, and was able to print the budget. - Joe's mobile device used Eve's device as a **proxy**, and thus Joe did **not need to provide a username and password**.
After a user has shared a resource, there may be a need to **monitor other users' usage of the shared resource** (for instance for charging them for usage). There may also be a need **controlling the usage of the shared resource** (for instance limit how many users that can use the resource at a time).	(Sharing did not exist in the ProblemScenario)	- Eve is automatically charged for her printouts because the library monitors her usage of the printer. - When Joe prints his budget on the shared library printer, Eve is charged, because **her device works as a proxy**.

Table 4.1: Connecting problems with their solutions.

4.2 Continuous and unpredictable changes in the ubiquitous environment

4.2.1 Retrieving appropriate software for utilizing a resource

Problem summary

Section 2.2.1: Users discover new resources when they move around in a ubiquitous environment. In order to utilize new resources, new software is needed. How can users easily acquire appropriate software?

Related functional requirements:

R-2, AppropriateSoftwareRetrieval
R-2.1, ListOfRetrievableSoftware
R-2.1.1, ContextAwareList
R-2.1.2, PredictionSortedList
R-2.2, NoSoftwareExists

GUI sketches

Figure 4.1, Figure 4.2.

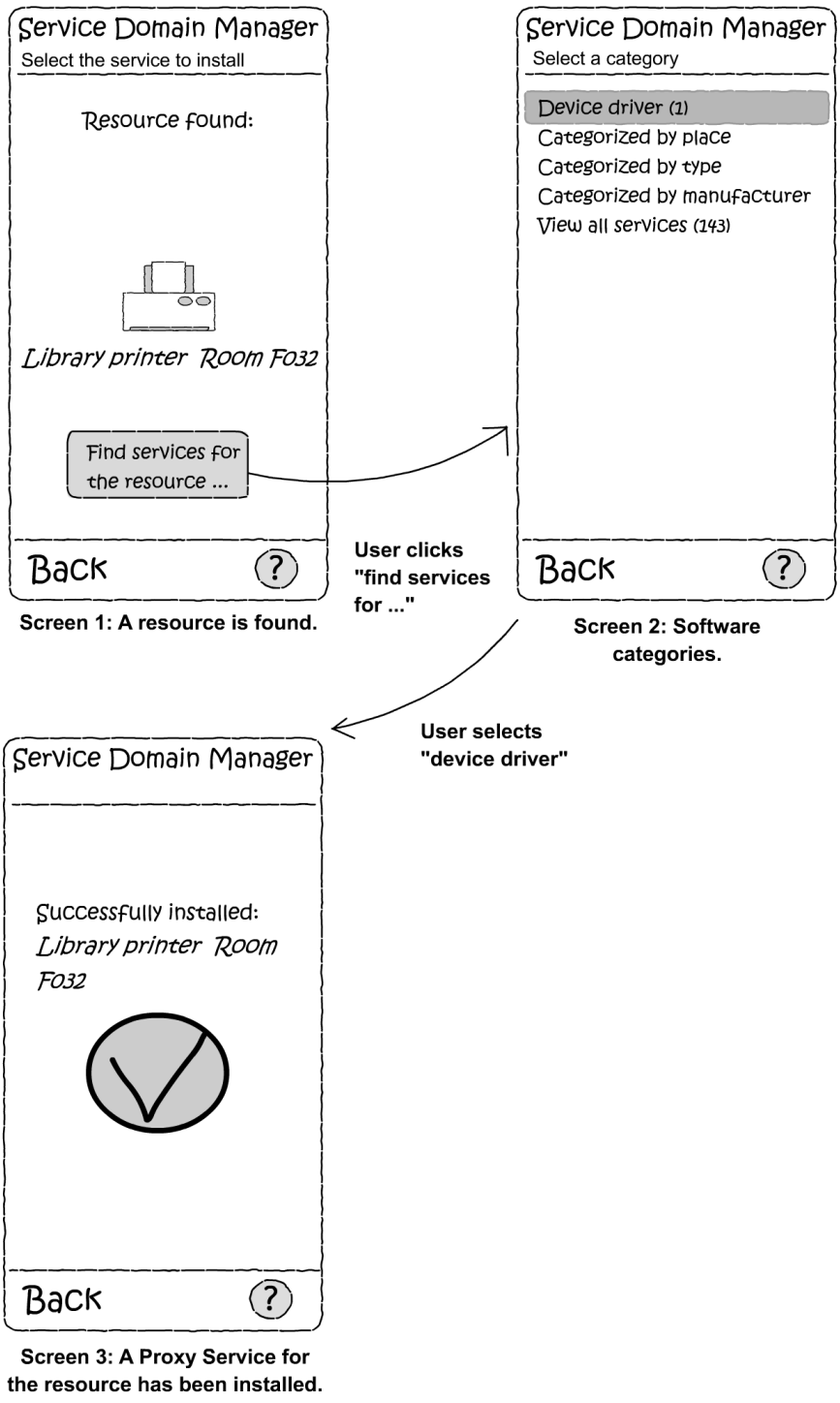


Figure 4.1: Installing a resource.

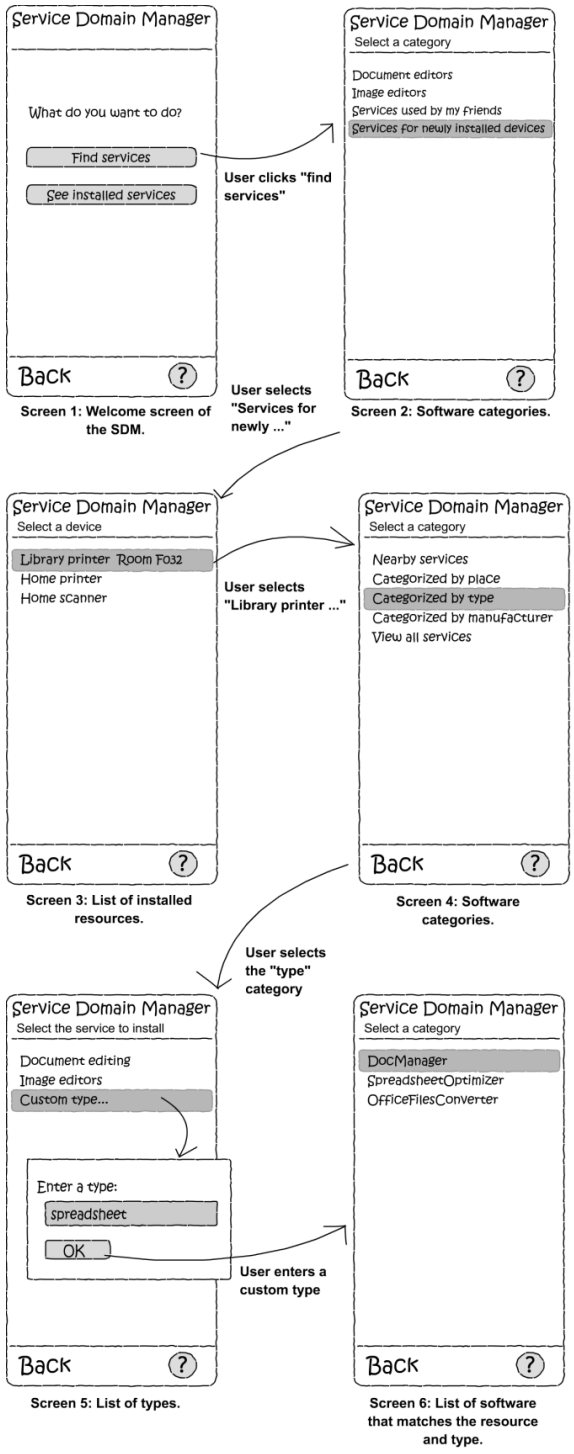


Figure 4.2: Finding Proxy Services that matches some criteria.

Solution
There are two ways to retrieve suitable Proxy Services. The first way is shown in Figure 4.1, the other in Figure 4.2:

- In Figure 4.1, a resource has been discovered by the UbiNode, and the user can find Proxy Services based on that exact resource. Only services that are suitable for the discovered resource is shown. This is done by keeping a registry of mappings between resources and Proxy Services. When a user selects a Proxy Service from the list, it is installed to the user's UbiNode (Figure 4.1, screen 3).
- In Figure 4.2, the user actively chooses to find Proxy Services, and is able to navigate through various categories to find what she is looking for.

When the user opens a category, Proxy Services and possibly additional sub-categories are listed. Categories can be based on various attributes, for instance the user's location. The ordering of the Proxy Services and the categories should be based on predictions of what the needs or wants.

Solution rationale
The basic idea of letting the user browse through suitable Proxy Services, was inspired by the way AppStore and Android Market allow users to download applications, as described in section 3.2.1.4.

This solution solves the problem of retrieving software suitable for a certain resource. The SDM shows only Proxy Services that are able to utilize the software in some way. This approach is similar to Windows' solution of only showing applications that supports a certain file extension (section 3.2.1.5). It is also similar to the approach of Jung et al. [14], which retrieves a software driver based on a bio-sensor device. However, in their approach, a device always pointed to one software driver, whereas a resource discovered in Ubicollab may point to many Proxy Services.

When a user discovers a resource, the solution solves the problem of long search time for finding suitable software, in the following ways:

- Instead that user spends a lot of time choosing search terms in a search site (either a general, like Google, or a specific, like

download.com) in order to find suitable software for a resource, SDM automatically shows the suitable software (Proxy Services) that matches and supports the resource.
- If SDM lists no Proxy Services, this immediately indicates that no software for the resource was found.
- The name of the Proxy Service does not matter, because the resource itself is used as a search key for finding appropriate Proxy Services, not a textual name.
- Fourth, SDM predicts which Proxy Service best matches the user's intensions, so the most relevant results will be shown first. This idea is taken from the AppRecommender (section 3.2.1.1).

When a user attempts to find software in general, that is, not just when discovering a resource, the problem of long search time is solved by using well-chosen categories. This idea was inspired by application listing systems such as Android Market, AppStore and Ubuntu Software Center. The utility of this solution is based on a good choice of categories. Some category suggestions are:

- Using categories based on attributes from the user's usage history. Location and time were suggested by Matsumoto et al. [22] as good attributes.
- A category for Proxy Services for resources that are physically close to the user, or similar context attributes.
- A category for Proxy Services that are shared by other users (explained later in this chapter).
- A category for Proxy Services used the last hour.
- A category for Proxy Services tagged with the text "office".
- A category for Proxy Services that represents resources on the web.

The user's problem of choosing the appropriate software when there are several choices are solved by:

- Predicting the user's intensions, as suggested by Kamisaka et al. [23].
- Having categories as explained above, making it easier for the user to browse the Proxy Services.

4.2.2 Managing existing software

Problem summary

Section 2.2.2: After installing software components, new issues arise. These issues are related to increased resource consumption of the user's mobile device and user-friendliness when managing many software components.

Related functional requirements

R-3, ControlResConsumption
R-4, ListSoftwareOrganized

GUI sketches

See Figure 4.3.

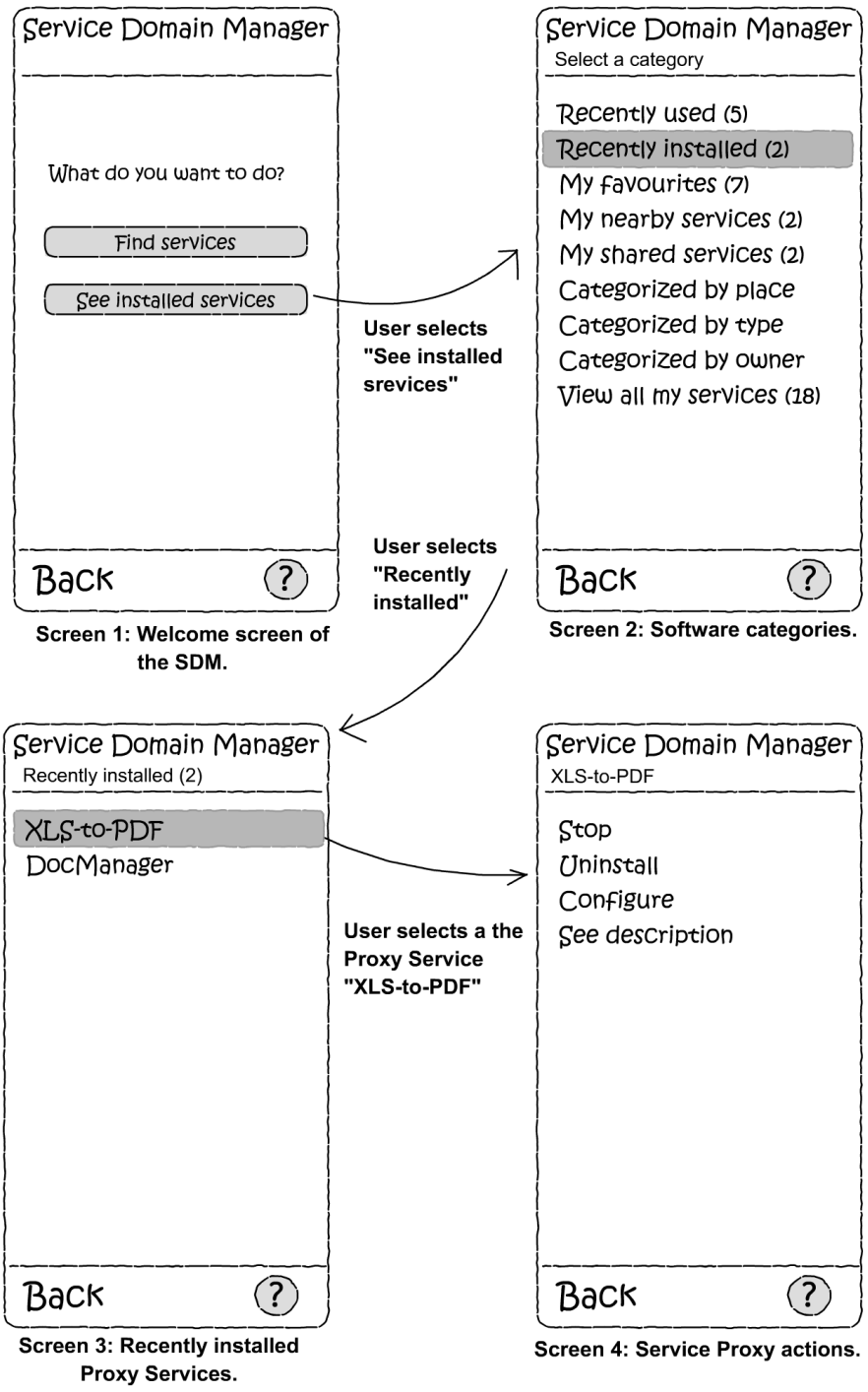


Figure 4.3: Managing Proxy Services.

Solution

In order for the user to manage installed Proxy Services (the software), the user needs a way to access them. This is solved by using a user interface that works the same way as described in section 4.2.1. The user accesses Proxy Services by navigating through the categories (screen 2), and then selecting the wanted Proxy Service (screen 3). When the wanted Proxy Service has been found, the user can "open" it and get a list of actions that can be executed on the Proxy Service (screen 4). These actions are:

- Start/Stop: If started, the Proxy Service can be stopped, and vice versa. Stopping a Proxy Service means that it's execution stops and it's functionality is disabled, and vice versa for starting. Because a Proxy Service is implemented using an OSGi *bundle*, this action includes starting/stopping the Proxy Service's bundle. See [26] for the exact definition of these actions.
- Uninstall: Uninstalling the Proxy Service means that it is removed from the UbiNode. This action implies uninstalling the bundle the Proxy Service resides in (explained in section 3.1.4).
- Configure: A Proxy Service may be configured by the user. What settings the user can change is up to the Proxy Service itself. This action exists only so that the user can access the Proxy Service's configuration.

Solution rationale

The user interface for Proxy Service (i.e. software) management works, as mentioned above, the same way as described in the section 4.2.1. This means that they are separate user interfaces, because Proxy Services available for retrieval and installation should be separated from installed Proxy Services. However, the inner workings of the user interfaces are the same, meaning that functionality such as categories and prediction of user intentions is also included in the user interface for managing Proxy Services.

The reason for using the an interface with the same functionality is that the user interface offers a good way for browsing and finding suitable Proxy Services. The categories and user intent prediction may be a bit different, but the basic functionality of user interface should still be the same. In this way, the problem of providing a user-friendly management system is solved.

The problem with increased resource consumption when more and more Proxy Services are installed, is solved by:

- Enabling the user to stop or uninstall Proxy Services. Stopped Proxy Services cannot do anything except exist, so stopping a Proxy Service reduces usage of the UbiNode's resources such as the CPU, storage, bandwidth and battery. Uninstalling a Proxy Service naturally includes the same effects.
- Enabling the user to configure Proxy Services. The Proxy Service's resource consumption may depend on its configuration, and by letting the user change the configuration, the resource consumption can be controlled. For instance, a Proxy Service for streaming music can include a setting for setting the maximum download bit rate. Conclusively, this option does not necessarily mean that resource consumption is reduced, but it enables this possibility.

The functionality described in the first point above was inspired by the functionality available for bundles in the OSGi framework ("OSGi bundle management" under section 3.2.2). Because Proxy Services are built on top of bundles, it is a natural choice to utilize the capabilities of bundles, because these capabilities solves our challenge with resource consumption control, and no additional solutions are required as they are already supported by OSGi. Further, Proxy Services do have some things in common with services in Android and in Windows (section 3.2.2), as both Proxy Services and services are able to run continuously in the background (does not have a UI), can offer functionality to other services, and they do use hardware resources. Because of this similarity, actions available on services could maybe be used on Proxy Services as well, such as starting and stopping. Windows does offer a lot of other functionality that can potentially could have been made available on a Proxy Service, such as blocking network access, updating, setting CPU priority, and more. However, this kind of functionality only enables more fine-grained adjustments of resource usage, and is not critical for resource control, and is thus not included in this solution.

Configuring Proxy Services, as described by point two above, works the same way as configuring services on Google Android (section 3.2.2) - the service

itself is responsible for what configuration is available. The user interface of SDM just provides the possibility to access and change this configuration.

4.3 Sharing of resources

Problem summary

Section 2.3: When users collaborate, there may be a need for sharing resources or functionality between their mobile devices (for instance a nearby projector, or some file converter functionality). Users utilizing a shared resource, may also need to use the resource on behalf of the user sharing it.

After a user has shared a resource, there may be a need to monitor other users' usage of the resource (for instance for charging them for usage). There may also be a need controlling the usage of the shared resource (for instance limit how many users that can use the resource at a time).

Related functional requirements

R-5, ShareSoftwareComponents
R-5.1, UseComponent-OnBehalf
R-5.2, MonitorShared-Usage
R-5.3, ControlShared-Usage

GUI sketches

See Figure 4.4.

Solution

The user interface works as follows: A user can get a list of actions available for a Proxy Service by selecting it and "open" it, as described in 4.2.2 (screen 1 and 2). An action for sharing the Proxy Service is added to this list (screen 3), so that the user can issue this action on a Proxy Service (that supports sharing). In other words, a resource is shared by sharing a Proxy Service. Other users will interact with the resource through this shared Proxy Service.

Issuing the action "Share service" will open a new screen (screen 4), where the user can add one or more Collaboration Instances (CI) the user wants to share the resource with (the result of adding a CI is shown in screen 6). The user may also specify a username and password that can be used to access the shared resource (screen 5).

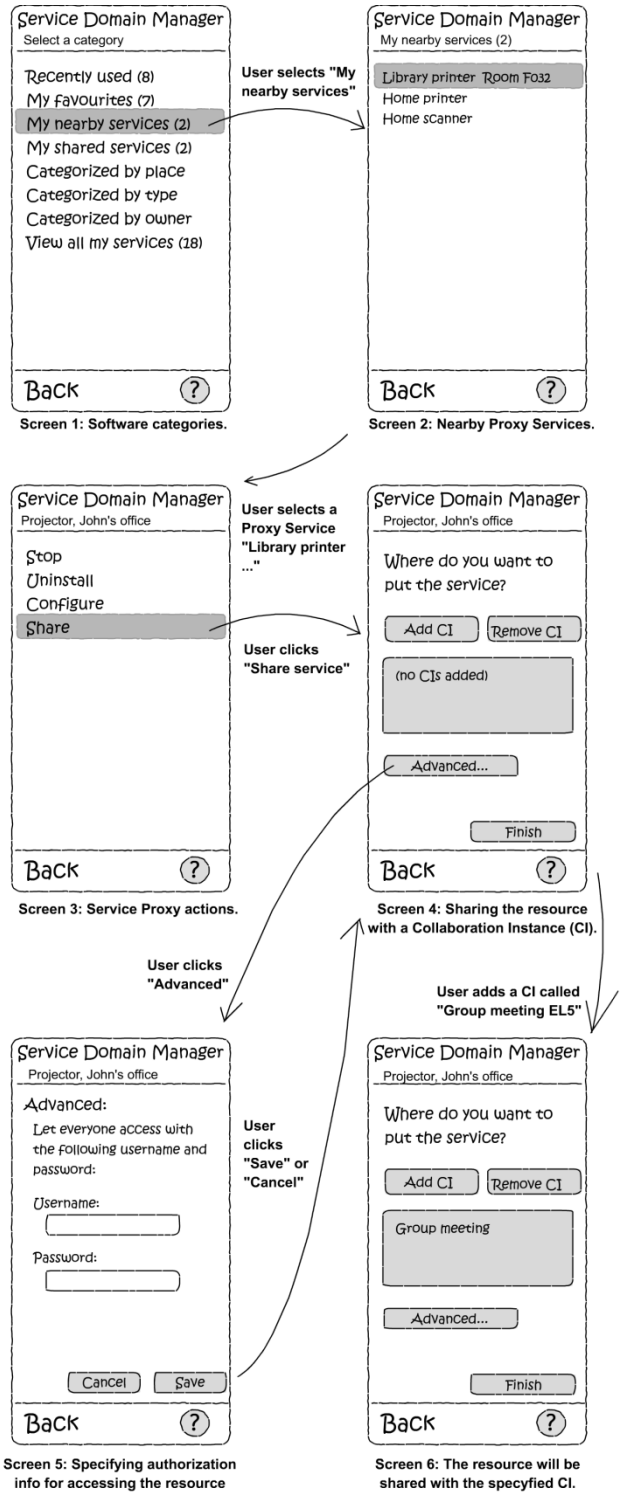


Figure 4.4: The user shares a Proxy Service.

When the user hits “Finish” (or similar), the UbiNodes of the other users residing in the selected CIs will get a notification about the shared resource.

If User A wants to utilize a Proxy Service shared by User B, User A will use the UbiNode’s resource discovery functionality in order to find the shared Proxy Services, just as with any other resource. This means that User A will use the installation procedure described in section 4.2.1 in order to find a suitable Proxy Service that can utilize the resource. Even if the resource is shared by another user, user A will need software that can interact with the resource (like a device driver), as explained in section 2.2.1.

The last paragraphs mainly described the user interface of the solution, but some additional explanation of the internals of the solution is required in order to get a sufficient understanding of it: Sharing of resources are done by sharing the Proxy Service(s) that utilizes the resource. Other users utilize the resource by communicating with the shared Proxy Service. In this way, the user can share not only resources, but also functionality that a Proxy Service implements (for instance a spreadsheet-to-PDF converter).

Solution rationale

The solution above solves the main problem of enabling a user to share a resource (through a Proxy Service), so that other users can utilize that resource. The solution also solves the problem of sharing functionality. This is made possible by the fact that Proxy Services are software components that offers functionality, and Proxy Services can be shared.

The problem of monitoring and controlling usage of the resource is solved by the fact that all communication with the resource goes through a proxy. This proxy can monitor the communication, and control usage by restricting what communication that can pass. Or, it can pass this information to a Proxy Services that handles monitoring and controlling the proxy.

A Proxy Service for monitoring and controlling usage can possibly also implement a user interface that shows this information to the user. This user interface could be a part of the SDM, and show information like “Eve and John are currently using this resource”, and give the user possibly actions like “Set maximum number of users”, and “Set time limit for sharing this resource”. However, such user interfaces are not suggested as part of this solution, as the

main focus is on sharing resources. Still, third party Proxy Services may implement this kind of functionality.

The basic idea of letting the user share resources and functionality basically came up from the problem description itself (section 2.3), but seeing that there are existing, working solutions like MobiShare (section 3.3.1), suggests that this is a viable approach. The solution presented here is not as advanced as MobiShare, but the basic ability of sharing is still covered.

Chapter 5 Implementation

This chapter describes the implementation of a system for managing Proxy Services on a user's UbiNode. The system is named *Service Domain Manager* (SDM).

Note that the implementation does not fully cover all the features of the proposed solution from Chapter 4. This will be evaluated later in this thesis.

The sections in this chapter is structured in the following way:

- Section 5.1 describes what existing systems the SDM will work further on.
- Section 5.2 states the method used for construction of the SDM.
- Section 5.3 describes the selection of platform components.
- Section 5.4 explains the architecture of the SDM.
- Section 5.5 details the components that constitute the SDM.

5.1 Starting point

As mentioned in section 1.2, ”Research context”, this thesis continues the work of the Ubicollab project, and specifically the works of Johansen [3] and Mora [4]. While most of their code was used for getting an understanding of the inner mechanisms of Ubicollab, Johansen implemented an earlier version of the Service Domain Manager (SDM). Mora made a few updates to this implementation, and this update was used as the initial implementation. The reason for using this initial implementation, was that it contained fundamental capabilities for service management, such as functionality for installing a Proxy Service.

5.2 Construction method

The construction method used in implementation phase is illustrated in Figure 5.1. Hevner et al. [5] emphasizes that the construction method of a design artifact must be rigorous. Our construction method is rigorous because of the use of scenario-driven and UML-based development.

5.2.1.1 Method steps explained

The construction method of the SDM is illustrated in Figure 5.1. The steps involved are described here:

- "SolutionScenario" (on the top): Refers to the scenario presented in section 4.1.
- Step 1: By "part scenario", we mean a part of the SolutionScenario that we want to create a solution for.
- Step 2: We considered solutions to how the problem(s) in the part scenario could be solved, and the most suitable solution was selected.
- Step 3: GUIs for the solution were sketched (if a GUI was part of the solution).
- Step 4: A minimum solution was coded to check if the solution from step 2 was possible to do. By "core features", we mean essential parts of the code that were required to work in order for the whole solution to work.
- Step 5: We created UML sequence diagrams in the creative process of designing a satisfiable solution (not simply after designing the solution).
- Step 6: The UML was implemented to create a solution. The solution was tested for correctness using whatever test methods seemed appropriate at the time. Further, the code was continuously "cleaned up" (refactored) to ensure its quality[5].
- Step 7: JUnit tests were made to verify that the code was working as it should, which made later changes to the code a more fail-safe process. However, this step was only done when creating the core part of the SDM.

Some criticism to step 7 must be pointed out: It is difficult to ensure the correctness and validity for a JUnit test that is created *after* the code it tests. A

[5] By "quality code", we mean code that conforms to well-known code-principles for clean code design, such as high cohesion, low coupling, a high degree of readability, and the single responsible principle, amongst other things.

better approach is using Test Driven Development (TDD) [27]. However, TDD was not used as construction method because of:

1. Time consuming problems with getting JUnit to work in OSGi.
2. TDD results in slower implementation progress for projects of the size of the SDM.
3. The SDM is a proof-of-concept application that does not need the code quality TDD aims to produce. Step 6 included testing solutions, and was considered to give a satisfactory level of correctness and validity of the SDM's source code.

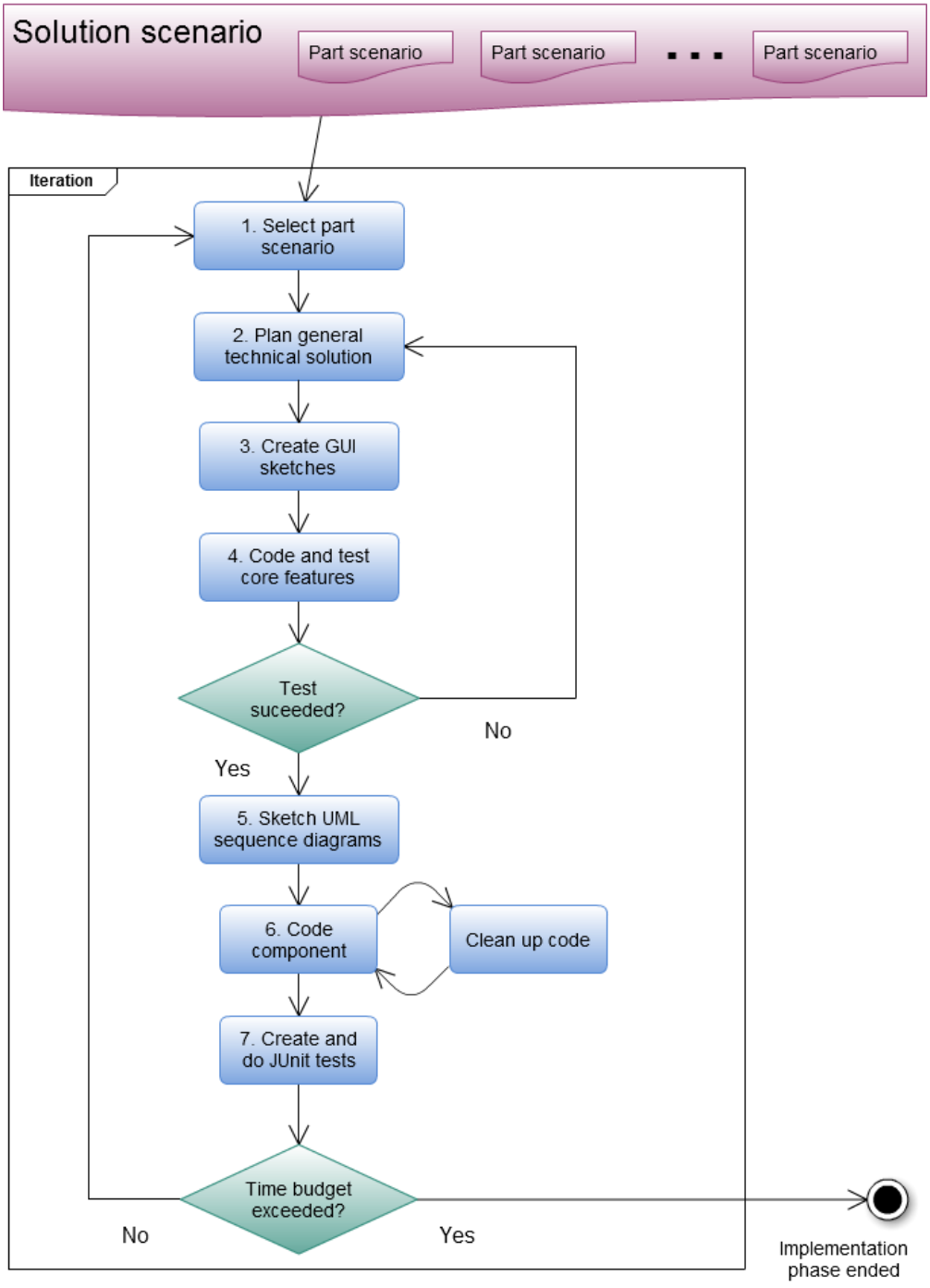


Figure 5.1: Construction method.

5.3 Platform choices

5.3.1 Selecting mobile operating system

Which kind of mobile device should we implement the SDM and Ubicollab on? The existing Ubicollab implementation runs on top of OSGi (as mentioned in section 3.1.4). OSGi can run on any Java specification [28], but this does not mean that Ubicollab can run on any Java based device. There are numerous implementations and versions of both OSGi and Java [29], and Ubicollab may not run or function properly on all of these. The exact platform requirements for the Ubicollab platform is of today not decided, and is not in focus of this thesis.

Therefore, a mobile OS that already had a working and running Ubicollab implementation was selected, namely Google's Android OS, which runs on several mobile phones. Android is an increasingly popular platform [30], and is a developer-friendly in that it is an open platform [31].

Ubicollab also runs properly on Windows Mobile Professional 6.1. However, this OS will in some future be replaced by Windows Mobile 7, which is not backwards compatible with earlier versions (6.x and below) [32]. Developing Ubicollab on a platform that will become obsolete in its next major version release should be avoided, because the implementation then must be ported.

5.3.2 Selecting OSGi implementation

There also exists a complete software developer kit (SDK) for implementing OSGi applications on Android, named ProSyst [33]. ProSyst offers integration with Eclipse [34], a powerful and popular software developer tool. Apache Felix [35] is another OSGi implementation for Android, but ProSyst appears to be the simplest to use, and has therefore been chosen.

5.3.3 Selecting Java version

The latest Java platform version supported by ProSyst is J2SE 1.5, and is thus the Java version used in the implementation.

5.3.4 Selecting GUI technology

The GUI of the SDM was chosen to be HTML-based, generated from Java Servlets, described further below.

The alternatives considered for GUI implementation was:

- Using an HTML-based GUI
- eSWT [36]
- Java Swing [37]
- Android's native GUI framework [38].

Criteria for choosing a suitable GUI was usability, platform-independence, and simplicity for making the GUI framework run on the ProSyst OSGi runtime.

5.3.4.1 An HTML-based GUI

There were three reasons for choosing an HTML-based GUI:

- HTML is platform-independent.
- Most smart phones are able view a HTML-based GUI. For instance, Android uses the WebKit API [39], while iPhone uses the UIWebView API [40].
- Using Java Servlets [41] enable dynamic creation of a GUI, similar to a scripting language like PHP [42] or JSP [43][6]. Additionally, ProSyst OSGi supports Java Servlets.

The HTML-based GUI does not support using native GUI elements, but this was outweighed by the reasons listed above. See REF _Ref266972134 \h Figure 5.7 for a screenshot of the HTML GUI for the SDM.

5.3.4.2 eSWT

Mora [4] uses eSWT for implementing the GUI for the resource discovery process in Ubicollab. eSWT enables a developer to create a GUI that can run on multiple mobile platforms while using native GUI elements when possible. This eliminates the need for porting the GUI to each platform in order to use native GUI elements.

eSWT was chosen for in earlier works of Ubicollab, because it enables the users to interact with native GUI elements, which supports usability [4]. However, eSWT was not used in SDM because it could not be set up and run successfully in the OSGi runtime on Android. More effort could maybe have solved this, but this project had limited resources, and the HTML-based approach offered a simple and sufficient alternative.

[6] So why weren't PHP, JSP, or other scripting languages chosen? JSP was considered, but we could not get it to work, and Java Servlets worked satisfactory.

5.3.4.3 Java Swing

Java Swing was not chosen for implementing GUIs in the SDM, because of two reasons. First, Swing does not support the use of native GUI elements [4]. Second, as for eSWT, Swing could not be set up and run successfully in the OSGi runtime on Android, while keeping the time budget.

5.3.4.4 Android's native GUI framework

Android's native GUI framework was tested by creating some simple GUI menus, and these GUIs supported the needs for the SDM. The created GUIs were difficult to run on the ProSyst's OSGi runtime, and were therefore not selected as a GUI. Because of time budget constraints and the fact that this approach was platform-dependent, this option was not further investigated.

5.3.5 Summary: Implementation stack

Figure 5.2 illustrates the final platform. The implementation of SDM was never tested on a real device during this work, which explains the "(Not tested.)" in Figure 5.2.

The architecture of Ubicollab is shown in Figure C.1 (on page 106), and the architecture of the SDM and its related components is shown in Figure 5.3.

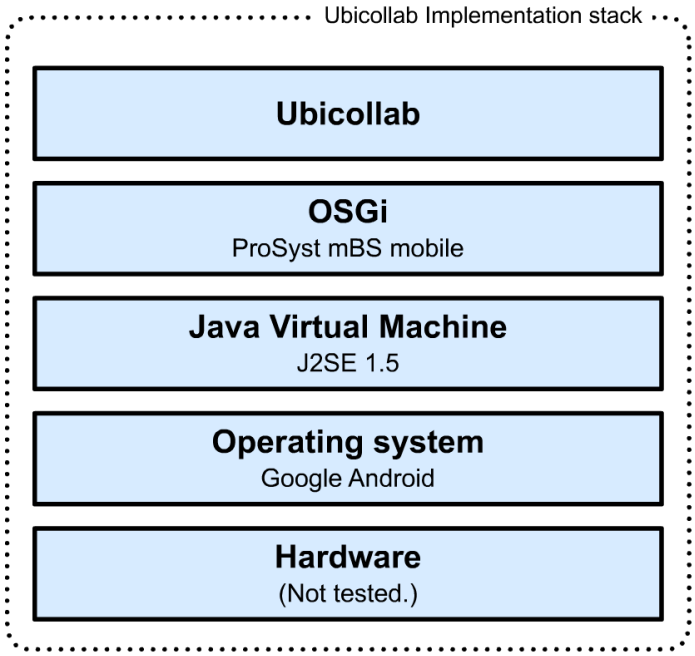


Figure 5.2: The implementation stack of Ubicollab.

5.4 Architecture

The following sub sections describes the components of the SDM. Figure 5.3 shows the OSGi bundles that constitutes the SDM.

We add some explanations to the figure:

- Package names have been simplified. See Appendix C.2 for a mapping of the names in Figure 5.3 and real Java package names.
- "JavaxServlet", "ProSyst OSGi for Android" and "JUnit" are external packages (in blue color), not implemented in this work. The rest are Java packages implemented as OSGi bundles, which together constitutes the SDM.
- Arrows denote dependencies between Java packages. For instance, the ViewServlet package is dependent on the JavaxServlet package.

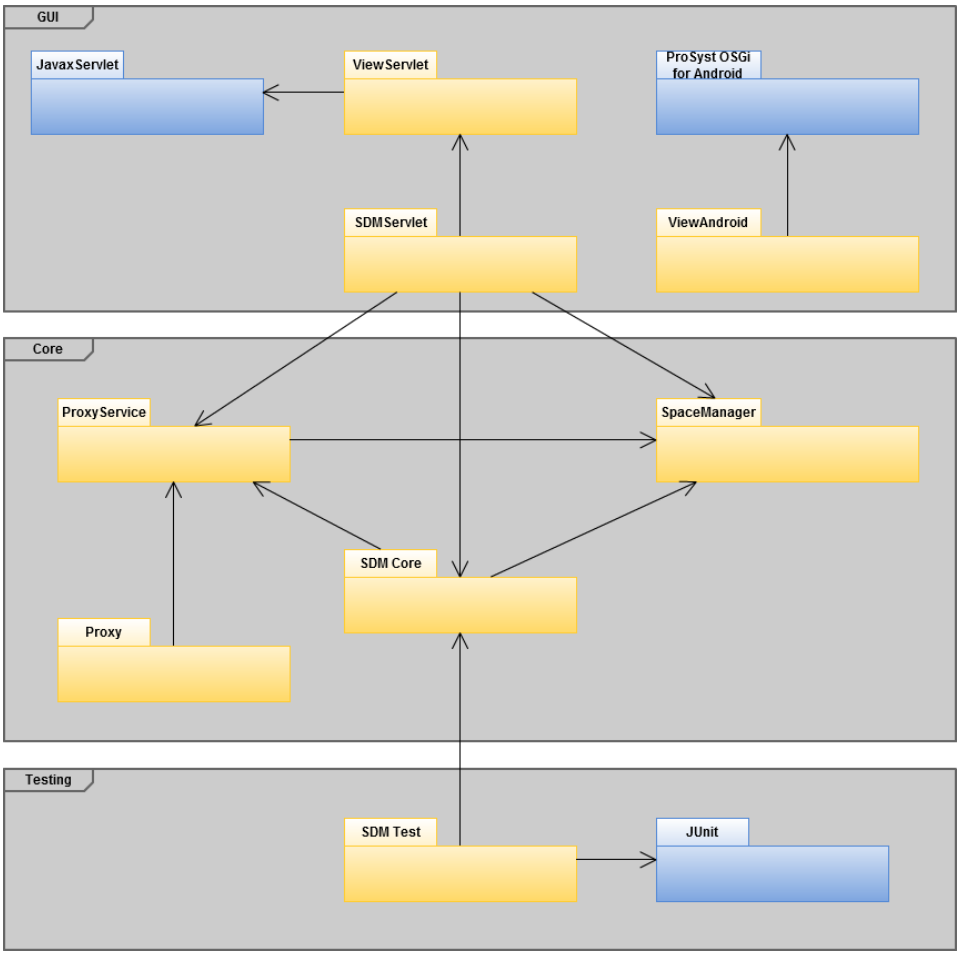


Figure 5.3: Internal architecture.

5.5 Components

This section describes the components of importance that the implemented solution consists of. Note that figures of components and APIs have been simplified for readability and intuitiveness.

5.5.1 GUI

5.5.1.1 ViewAndroid

SDM uses a HTML-based GUI, see an example in Figure 5.7. The user browses the GUI through a Android WebView-component [44] that show can show HTML pages. This component is named "ViewAndroid" in Figure 5.3.

5.5.1.2 ViewServlet

The HTML pages are hosted by a Java Servlet [41], which is managed through a bundle called *ViewServlet*. ViewServlet can host web pages, using Java Servlet functionality. It enables a client to access a HTML GUI through an URL. For instance, the GUI of the SDM is accessed through the URL *http://localhost:1149/sdm?page=MainMenu*.

The ViewServlet itself does not produce HTML pages, it fetches the appropriate HTML page from a HTML provider based on the incoming HTTP request. In the URL above, the HTML provider is SDMServlet, identified by the "sdm" part of the URL.

5.5.1.3 SDMServlet

SDMServlet creates the GUI of the SDM (see Figure 5.7). SDMServlet contains all the HTML pages of the SDM GUI. The content of a HTML page is returned whenever ViewServlet requests it.

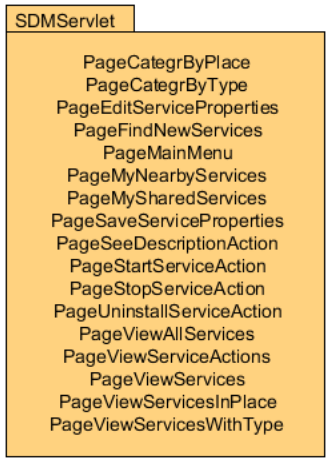


Figure 5.4: The SDMServlet package and its classes.

5.5.2 Service Domain Manager core

This section describes the core component of the SDM.

5.5.2.1 Screenshots

Note that the GUI uses the term *service* for Proxy Service. Arrows indicate a connection between a user clicking a link in the GUI, and the resulting screen.

Figure 5.5: The startup screen on Android OS.

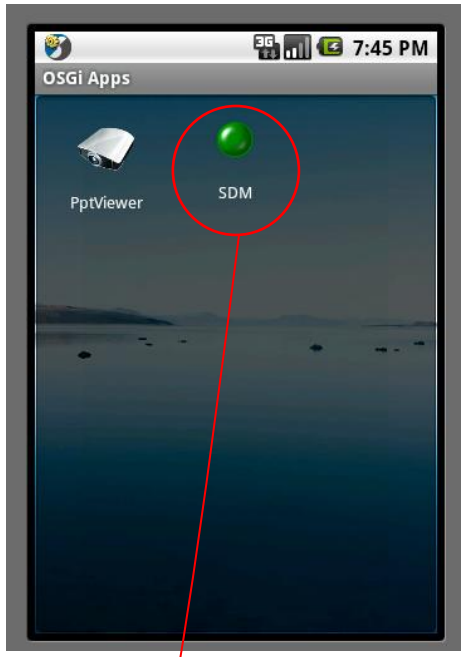


Figure 5.6: The OSGi application menu.

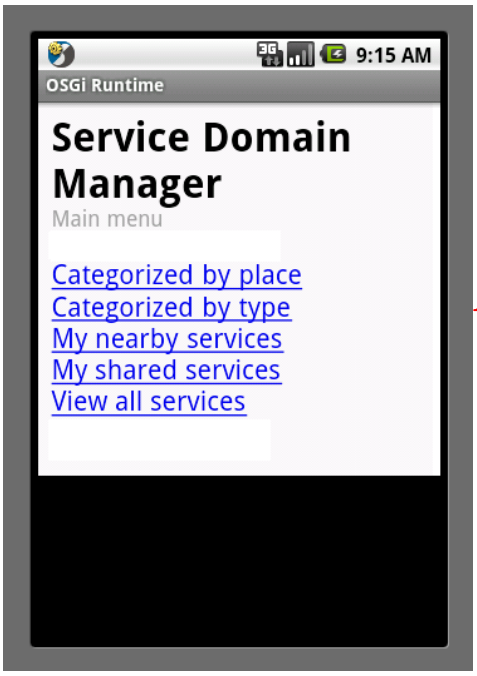
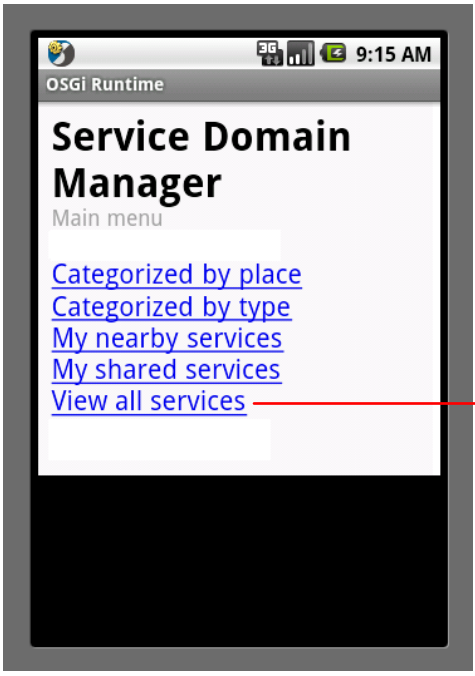


Figure 5.7: The main menu of Service Domain Manager.



(Continued from last page)

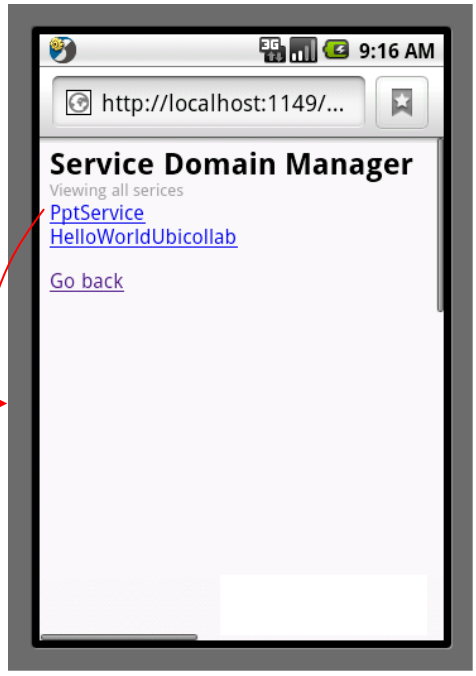


Figure 5.8: List of all installed Proxy Services.

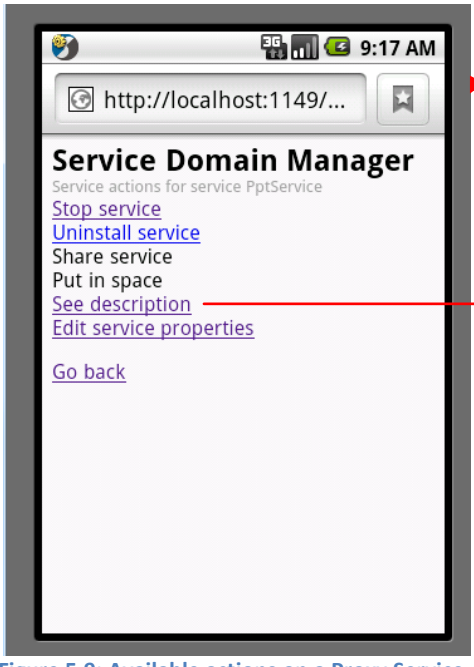


Figure 5.9: Available actions on a Proxy Service.

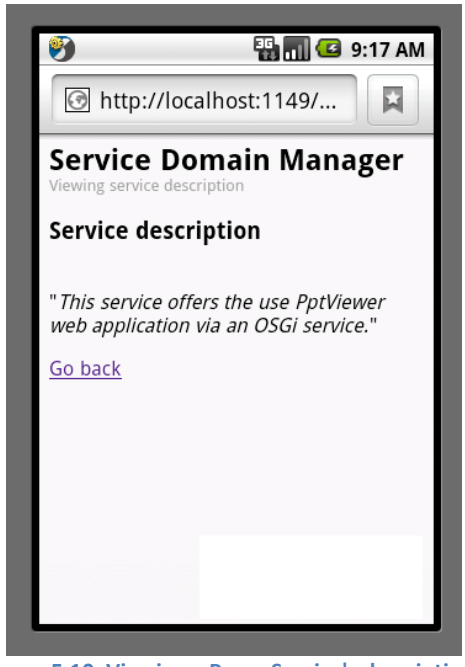


Figure 5.10: Viewing a Proxy Service's description.

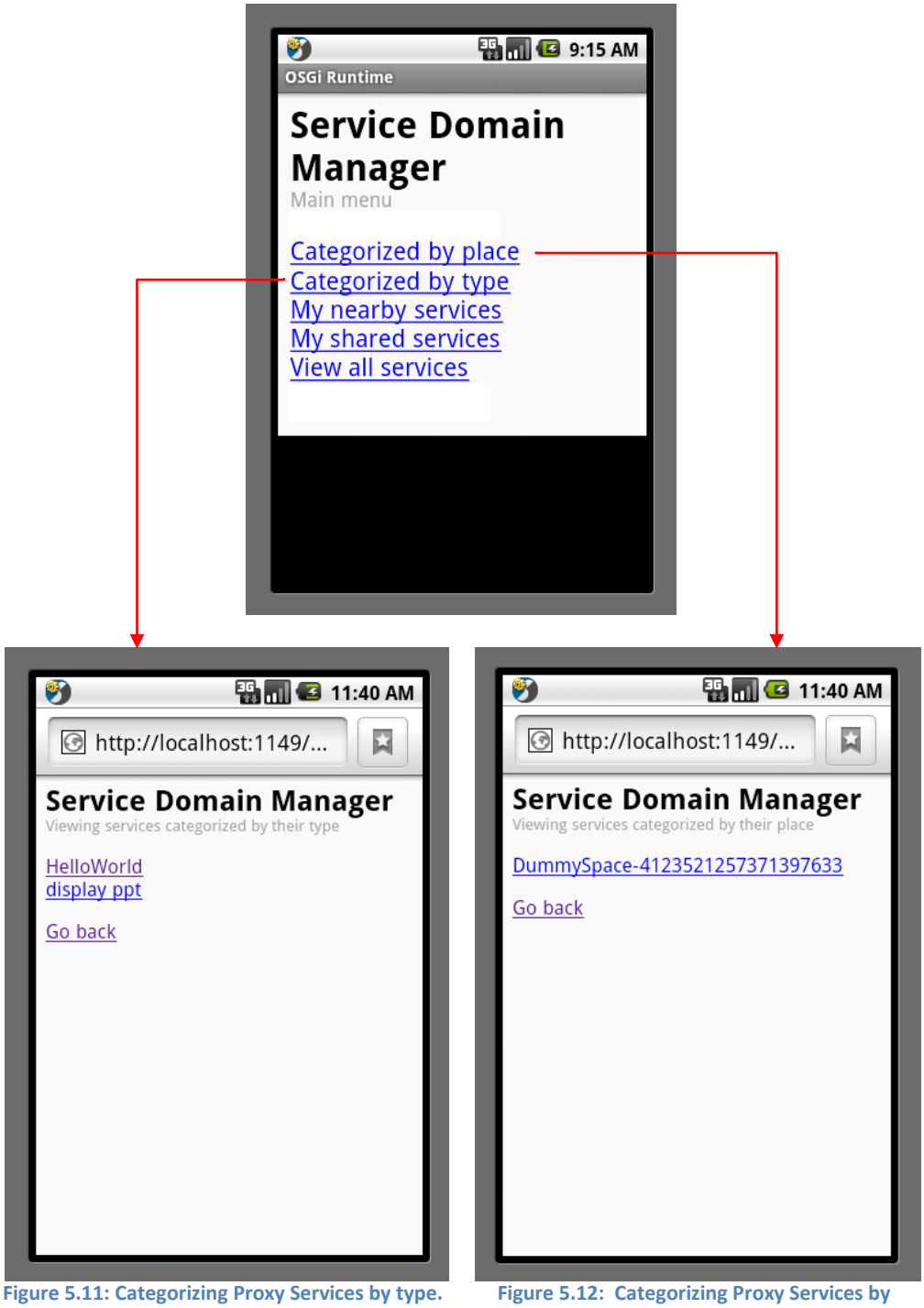


Figure 5.11: Categorizing Proxy Services by type.

Figure 5.12: Categorizing Proxy Services by Space.

5.5.2.2 Functionality

This section describes the capabilities of the implemented SDM, most of which are visible from the GUI, but some only available at the API level.

Functionality accessible from the GUI

Categorize the list of Proxy Services: The SDM is able to view a list of installed Proxy Services. The user can list all Proxy Services, or categorize them by place (Collaboration Space), nearby places or type, or view all her shared Proxy Services.

Manage Proxy Services: When the user accesses a Proxy Service in the GUI, she can execute one of the following actions (Figure 5.9):

- "Stop service": Stops the Proxy Service.
- "Uninstall service": Uninstalls the proxy.
- "Put in space": Puts the Proxy Service into a Collaboration Space. Not implemented into the GUI, but possible at the API level.
- "See description": Views the description of the Proxy Service.
- "Edit service properties": Lets the user configure the Proxy Service, see an example in Figure 5.19.

Internal functionality

Share service: A Proxy Service can be shared so that other users can utilize it. Sharing Proxy Services is not implemented into the GUI, but sharing of Proxy Services is still supported by the platform.

Install Proxy Service:

```
long installService(String url);
```

Listing 5.1: Interface method installService.

The method installs a Proxy Service from the given URL. The URL must point to a JAR file.

Activate Proxy Services in a space:

```
boolean setActiveSpace(
                    Space space, boolean stopOtherServices);
```

Listing 5.2: Interface method setActiveSpace.

The method starts all Proxy Services in the given Collaboration Space first parameter. If the boolean `stopOtherServices` is set to *true*, then all Proxy Services not in the given Collaboration Space is stopped.

Automatic detection of changes to Proxy Services:

SDM listens for changes to bundles in the OSGi frameworks, in order to detect when any Proxy Service is either installed, uninstalled, started, stopped or updated.

Figure 5.13: Interface for the component ServiceDomainManager.

5.5.3 Proxy Service

For a conceptual description of a Proxy Service, see the glossary section.

Proxy Services are implemented as OSGi bundles, but have in this implementation been given additional properties so that the Ubicollab platform

can separate them from regular OSGi bundles. These properties are described in Appendix C.2.

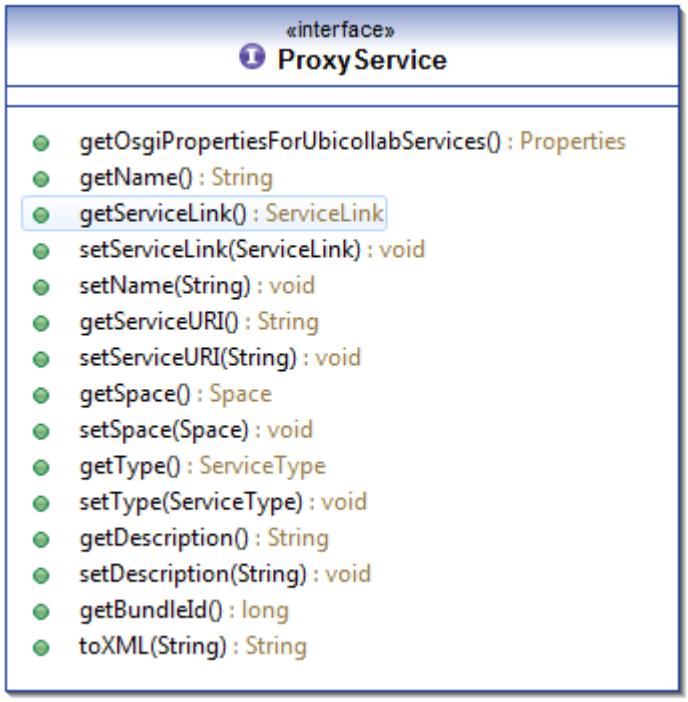


Figure 5.14: Interface for the ProxyService component.

5.5.3.1 ServiceSettings - configuring a Proxy Service

Proxy Services define their own configuration, that is, which fields the configuration should consist of, and their initial value. The user can change these values through the SDM's GUI. An example is shown in Figure 5.19. SDM generates this GUI based on the Proxy Service's configuration fields. The container for the configuration is named “ServiceSettings”, and is a part of the ProxyService component. The configuration values are stored by the ProxyService using XML.

The reasons for using this solution are the following:

- It allows users to configure Proxy Services.
- The user can access the configuration from one known place, that is, the menu shown in Figure 5.9.

- Proxy Services do not need to provide a GUI in order to be configurable.

5.5.4 Sharing of Proxy Services

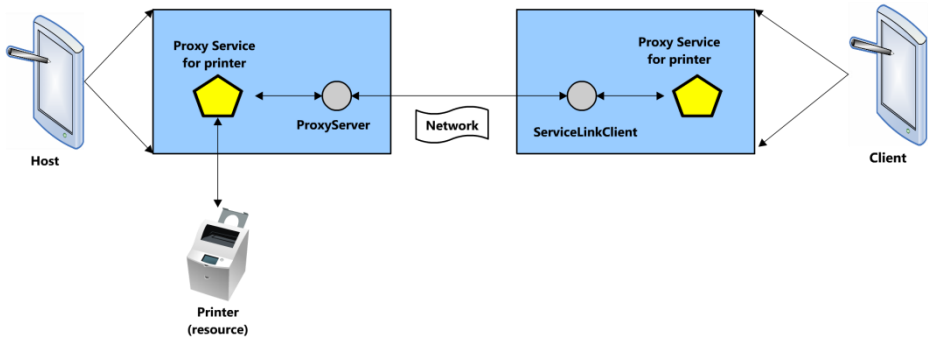


Figure 5.15: The internals for sharing a resource

Sharing of Proxy Services is illustrated in a simple way in Figure 5.15. The figure is simplified, and a more detailed view can be seen in Appendix C.5. We will also explain the communication process further here. We will name the client user "Client", and the host user "Host", as in the figure.

There are two internal components that make sharing of Proxy Services possible:

1. A Java Servlet implemented by a component named ProxyServer. This component is used by the host user's UbiNode.
2. A component named ServiceLinkClient. This component is used by the client user's UbiNode.

The Client sends messages to the Host by sending a HTTP request with the message to send as a GET-parameter. For instance, sending the message "`PrintHowAreYou`" can be done by opening the following URL:

```
http://<ip_of_Host>/printerProxy/?msg=PrintHowAreYou
```

The Host returns an answer through the HTTP-response received when opening the URL above. By opening the URL above in a regular web browser, one can directly see the response. For instance, the ProxyServer may return the simple text "`IAmFineThanks`".

Now that communication is in place, sharing of Proxy Services is in principle possible. The host and client application must use a common protocol for communication, as SDM does not address this. For instance, the host application of PptViewer (presented later in this chapter) only accepts message on the form "`openUrl http://...`".

5.5.4.1 ServiceLink

Using a Proxy Service "on behalf of" other users, as explained by requirement R-5.1, is implemented by making sure that all communication between a Host and its shared ProxyService goes through a ProxyServer, or an equivalent. In SDM, an interface named "ServiceLink" is used for classes that can take care of communication from both the host UbiNode, and a client UbiNode.

5.5.4.2 Discovery of shared Proxy Services

In order to let other users be able to discover the shared Proxy Service, the following method can be used: User A can find the message proxy if User B notifies all users in the Collaboration Instance (CI) about the shared Proxy Service, and this notification includes an URL to the message proxy. This has not been implemented in the current solution.

5.5.4.3 R-OSGi's incompatibleness with Android OS

Why was not R-OSGi used in our solution? Research and solution approaches described earlier in this thesis indicated that R-OSGi should be used.

The simple answer is that we could not get it to work. The problem was the following:

R-OSGi enables transparent sharing of OSGi services by interpreting the contents of Java ".class" files, that is, the byte code of classes. These class files normally reside in the OSGi bundle, that is, the bundle's JAR file.

On Android OS, uses its own format for JAR files. It uses a process called "dexifying" in order to optimize the JAR files. This process replaces all class files in a JAR file with a single file called "classes.dex" that the Android OS can read. As a result, R-OSGi cannot find the class files it needs. And conclusively, R-OSGi could not be used in our implementation.

During the work with R-OSGi, we came across possible solutions to this problem (see [45]), but limitations in the time budget denied further research.

5.5.5 Space Manager

The Collaboration Space Manager is a Platform Service in the Ubicollab platform, which is responsible for retrieving a user's current physical location, and managing Collaboration Spaces.

As no implementation existed when implementing the SDM, a stub ("dummy") implementation was used instead.

5.5.6 JUnit tests

The JUnit tests perform various tests on the SDM Core.

5.5.7 A demonstration application: PptViewer

The usage of of PptViewer is shown through a scenario.

5.5.7.1 Scenario

*Eve has rented the room "G128" for today's presentation, in which Joe and she are to talk about the their latest research. When renting the room, she got a username and password to a projector in the room, which is accessible on the wireless network. She has configured her UbiNode's presentation viewing application, PptViewer, to use that projector (***Figure 5.19***).*

*People are set, and she opens her presentation in PptViewer (***Figure 5.16***), and navigates through her slides (***Figure 5.17***) using her UbiNode as a remote controller to the projector. The projector shows the slides in a Firefox browser window (***Figure 5.18***). Soon Joe is up, but not equally well prepared, he asks her for help setting up the PptViewer on his UbiNode. When Eve tries to configure the PptViewer on Joe's UbiNode to use the room's projector, the UbiNode alerts "No access to resource!". Eve has forgotten the username and password she received for the projector, so instead, she uses her UbiNode to share the projector with Joe. Finally Joe is able to configure his PptViewer to use the shared projector instead of using it directly. He then opens his presentation in PptViewer, and holds his presentation.*

5.5.7.2 Screenshots

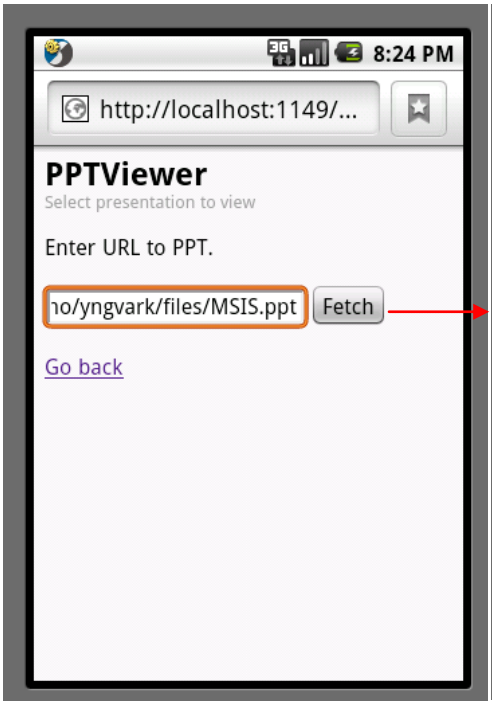


Figure 5.16: Opening a PPT presentation with PptViewer.

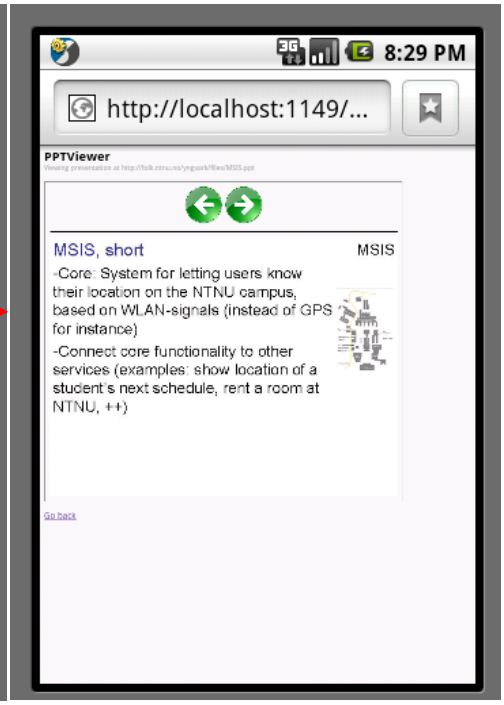


Figure 5.17: Viewing and controlling a PPT presentation from Android.

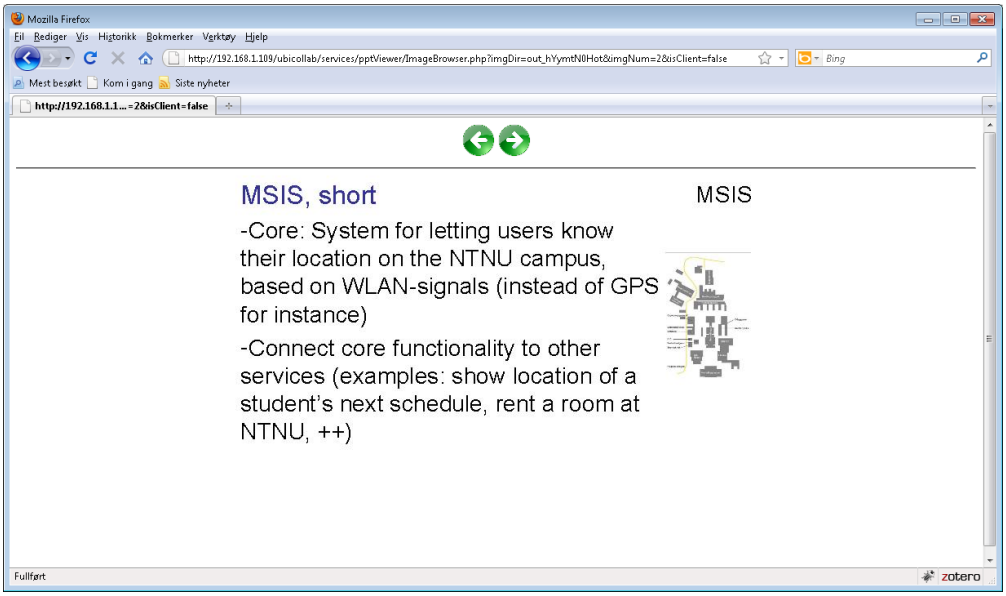


Figure 5.18: The server presentation application.

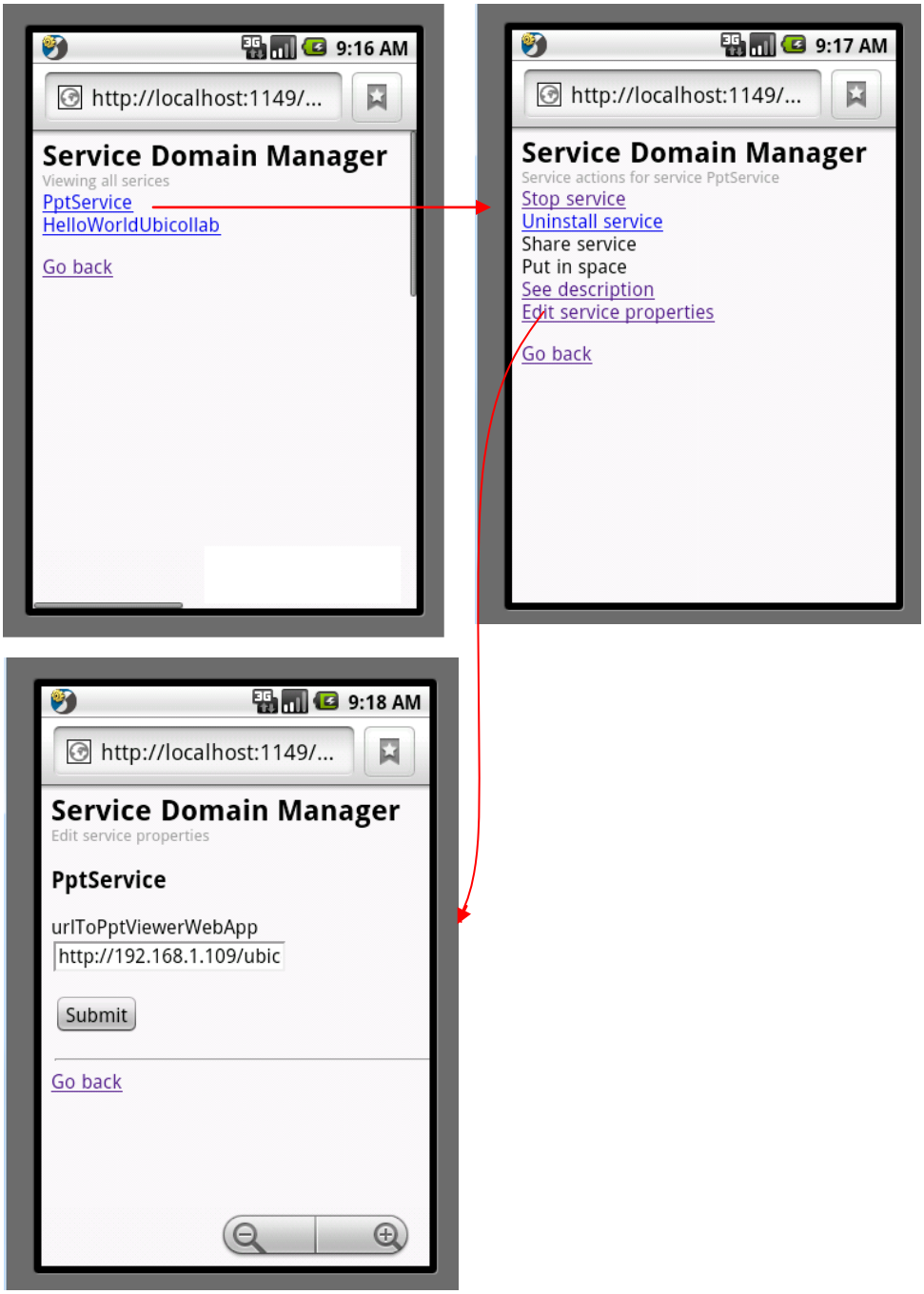


Figure 5.19: Editing service properties of the PptViewer.

5.5.8 Portability

The only component of the SDM that is platform dependent to Android, is the ViewAndroid component (section 5.5.1.1), as this uses the Android WebKit API. ViewAndroid is completely decoupled from the rest of the SDM, so if SDM are to be ported to another OS, only the ViewAndroid component should be needed to be replaced. We say *should*, because there may be additional changes required, due to the fact that OSGi implementations vary, even if they adhere to the same OSGi specification. For instance, the SDM was tested during this work on the Knopflerfish OSGi framework running on a desktop PC without succeeding.

Chapter 6 Evaluation and discussion

This chapter evaluates and discusses the completed work in this thesis. The sections are structured the following way:

- Section 6.1 states the methods used for evaluating the implementation described in Chapter 5.
- Sections 6.2 to 6.4 evaluate the implementation based on the methods described in section 6.1.
- Section 6.5 answers the research questions stated in section 1.3.
- Section 6.6 describes the limitations of the work in this thesis.

6.1 Methods used for evaluation

Several methods will be used for evaluating the implementation described in Chapter 5. These methods are:

1. Investigating the implementation's fulfillment of the requirements specification presented in section 2.4.
2. Investigating if and how the implementation supports the proposed solution presented in Chapter 4.
3. Evaluating the use of a real-world application, named PptViewer. The application utilizes the implementation's capabilities that enable users to share and configure resource controllers.

Executing method 1 will give an overview of the completeness of the implementation. Method 2 and 3 demonstrate the utility, quality and efficacy of the implementation.

Method 3 is also motivated by the work of Edwards et al. [46]. Edwards et al. argue that evaluating an infrastructure that aims to support a certain user experience, must be done by evaluating the user experience offered by applications that utilize the infrastructure. In our case, the solution implementation includes infrastructure that enables users to utilize resources

shared by other users, so we evaluate this part of the infrastructure by evaluating the use of PptViewer.

Note that this evaluation method only evaluates the parts of the infrastructure related to sharing of resources, and not the complete implementation.

The next sections present the evaluation based on the methods listed above.

6.2 Fulfillment of requirements specification

Table 6.1 shows how the SDM fulfills the requirements specification from section 2.4. It shows that the functions implemented were related to management and sharing of existing software components, while those functions related to the discovery of services were not or partially implemented.

Table abbreviations
Cp: Complexity
Pr: Priority
I: Implemented
PI: Partially implemented
NI: Not implemented

Requirement ID	How is the requirement fulfilled?	Cp	Pr	I	PI	NI
R-1, Mobiliity	The SDM is able to run on the ProSyst OSGi runtime, which are supported on the mobile platforms Windows Mobile and Google Android. The SDM has been tested on Google's Android Emulator. The SDM is able to run on the ProSyst OSGi runtime, which are supported on the mobile platforms Windows Mobile and Google Android. The SDM has been tested on Google's Android Emulator.	M	H	**X**		
R-2, Appropriate-	R-2 is not fulfilled, because the SDM has not been integrated	H	H		**X**	

Requirement ID	How is the requirement fulfilled?	Cp	Pr	I	PI	NI
SoftwareRetrieval	with the existing resource discovery system of Ubicollab, that is, the Resource Discovery Manager. Further, several of the capabilities described in section 4.2.1 have not been implemented. However, the SDM supports retrieval of appropriate software because it is possible to install external Proxy Services via the SDM's API.					
R-2.1, ListOf-Retrievable-Software	The SDM is able to list Proxy Services, and the user is able to select any Proxy Service from the list and get a menu of appropriate actions for that Proxy. However, the SDM only lists *installed* Proxy Services, that is, Proxy Services that resides on the user's UbiNode.	M	H			**X**
R-2.1.1, ContextAware List	The user can select "My nearby services", which lists Proxy Services that reside in the same Collaboration Space as the user. However, the Platform Service responsible for giving the user's current location has not been implemented yet (not part of this thesis). See comments in section 6.2.1 for details.	H	H		**X**	
R-2.1.2, Prediction-SortedList	The SDM does not include any means to predict user intents and needs.	H	M			**X**

Requirement ID	How is the requirement fulfilled?	Cp	Pr	I	PI	NI
R-2.2, NoSoftware-Exists	SDM shows software by using a list (such as in Figure 5.8). If this list is empty, we assume the user understands that no software is available.	M	M	**X**		
R-3, ControlRes-Consumption	The user is able to stop, uninstall and configure a Proxy Service (see Figure 5.9). Stopping and uninstalling frees up resources for the rest of the system. Configuring the Proxy Service allows the Proxy Service to include settings that can control its resource consumption.	M	M	**X**		
R-4, ListSoftware-Organized	The SDM is able to show software in different categories, as shown in figures in section 5.5.2.1.	M	H	**X**		
R-5, ShareSoftware-Components	Proxy Services can be shared and used by other users across the network, as described in section 5.5.4, and demonstrated by the PptViewer in section 6.4.	H	H	**X**		
R-5.1, UseComponent-OnBehalf	The demonstration of PptViewer (section 6.4) showed that one user could send requests to a web application through the UbiNode of another user.	H	M	**X**		
R-5.2, MonitorShared-Usage	It is possible to monitor all usage originating from other users by using the ServiceLink component, as described in section 5.5.4.1.	M	M	**X**		
R-5.3, ControlShared-Usage	This is not supported in the current implementation, but the ground work is done through the ProxyServer and ServiceLink components.	H	M			**X**
Total				**8**	**2**	**3**

Table 6.1: Fulfillment of requirements specification.

6.2.1 Comments to Table 6.1

A note to R-2.1.1

A Collaboration Space is currently not linked to a physical location, because the SpaceManager Platform Service of Ubicollab responsible for this was not implemented at the time making the SDM. In the current implementation, the user always reside in one default Collaboration Space (seen in Figure 5.12).

This requirement is considered partially implemented because the SDM's implementation is ready for utilizing the current location of the user, but is incomplete because of the missing functionality of another Platform Service, which is out of the scope of this thesis.

6.3 Fulfillment of the proposed solution

This section investigates if and how the implementation of the SDM fulfills the proposed solution presented in Chapter 4.

6.3.1 Scenario analysis

This section investigates the SDM's support for the challenges in the SolutionScenario presented in section 4.1.

This is done by first repeating a relevant part of the scenario, and then stating if and how the SDM solves or supports the situation described in the repeated part. **Bold text** is used to emphasize what parts of the repeated text is commented.

The implementation of the SDM, presented in Chapter 5, will be referred to here as the "SDM", and is not to be confused with the solution system in the scenario, which is referred to as the "*Service Domain Manager*", in italic.

6.3.1.1 The analysis

---

1. *She finds her UbiNode, and makes a search for printers on the wireless network.*

Implemented: The scenario suggests that the solution shall run on a UbiNode, which is a mobile device. The SDM is implemented on the Android OS, and is thus mobile.

2. *She finds one called "Library printer Room F032", selects "Find services for this resource", selects "Device driver (1)" , and is able to get a Proxy Service that her UbiNode needs for accessing the printer (***Figure 4.1***).*

Partly implemented: The SDM does not support finding and installing new Proxy Services. However, a GUI similar to the GUI in screen 2 in Figure 4.1 has been made, and shown in Figure 5.7. Also, the SDM supports installation of Proxy Services on the API level.

3. *Eve notices that she does not have any Proxy Service for opening and printing the budget, so using her UbiNode,* **she starts Service Domain Manager and selects** *"see installed services".*

Partly implemented: Starting the SDM is shown in Figure 5.5 and Figure 5.6. Figure 5.7 shows the startup screen of the SDM, which corresponds to screen 2 in Figure 4.3. However, the startup screen itself is missing (screen 1, Figure 4.3).

4. *She then chooses to list services for newly installed devices, which that are tagged with "spreadsheet". She installs the first on the list, "DocManager" (***Figure 4.2***).*

Not implemented: Listing Proxy Services for newly installed resources, the scenario intends that the solution should include a way to list Proxy Services that are associated with a specific resource. The SDM does not implement functionality for this.

Implemented: However, finding Proxy Services using tags are possible to some extent, but is called "type" in the SDM. Categorizing by type is shown in Figure 5.11. There is currently a limitation of one tag per Proxy Service in the SDM.

5. *She suspects one of the new Proxy Services to cause the problems, so she uses Service Domain Manager to get a list of recently installed Proxy Services (***Figure 4.3***).*

Not implemented: The SDM does not support listing of recently installed Proxy Services.

6. *Eve chooses XLS-To-PDF, and finds the configuration button in the menu as she is used to, and clicks it.*

Implemented: The SDM allows the user to configure Proxy Services. An example of this is shown in Figure 5.19.

7. *Eve checks the description of the Proxy Service "Library printer Room F032", and is able to find directions to Room F032, where the printer is located.*

Implemented: The SDM allows the user to see descriptions of a Proxy Services, as seen in Figure 5.10.

8. *While walking,* **Eve issues her printout and receives a message** *that says her credit card has been charged by a total of $1.40, $0.20 per page.*

Implemented: The component ServiceLink (part of the Proxy component) enables resource monitoring, as described in section 5.5.4.1.

9. *Joe wants to print some copies of the budget to hand over to the client at a later meeting, so Eve creates a Collaboration Instance named "Group Meeting".*

Out of scope: Collaboration Instances on the Ubicollab platform are managed by the Collaboration Instance Manager, which is not a part of the work in this thesis.

10. *She then shares the library printer with the* Group Meeting *Collaboration Instance, so Joe's UbiNode is able to find and install it (***Figure 4.4***)*

Partly implemented: Proxy Services can be shared and used by other users across the network, as described in section 5.5.4 and demonstrated by the PptViewer application in section 6.4. However, sharing is not implemented into the GUI.

11. *Joe prints his budgets via Eve's UbiNode, and Eve is notified that her credit card has been charged by a total of $1.20, $0.20 per page.*

Implemented: The ServiceLink (section 5.5.4.1) enables a UbiNode to utilize a shared Proxy Service on behalf of the host UbiNode.

6.3.2 Summary

Six capabilities were implemented, three were partially implemented, and two were not implemented. The crucial capability missing was that related to installing new Proxy Services.

6.4 Evaluation of a real world application: PptViewer

6.4.1 Scenario evaluation

We will here discuss how the capabilities of PptViewer were supported by the SDM. In other words, we evaluate the SDM through evaluating an application that utilizes its infrastructure and/or capabilities, as suggested by Edwards et al. [46]. This evaluation uses the scenario section 5.5.7.1 as basis, and we will refer to it as the PresentationScenario.

1. *She has* **configured** *her UbiNode's presentation viewing application, PptViewer, to use that projector (**Figure 5.19**).*

Implemented: Eve's ability to configure the application is supported by the configuration capability of the SDM.

2. *People are set, and she opens her presentation in PptViewer (**Figure 5.16**), and navigates through her slides (**Figure 5.17**)* **using her UbiNode as a remote controller to the projector**.

Implemented: When Eve opens her presentation in PptViewer, the PptViewer checks its configuration for the URL to the host web application that is the backbone of the application. The SDM enables the PptViewer to have its own configuration through a class called ServiceSettings (section 5.5.3.1).

3. *Soon Joe is up, but not equally well prepared, he asks her for help* **setting up the PptViewer on his UbiNode**.

Partly Implemented: There are many steps not mentioned here. If we assume that Joe already has the PptViewer installed, and only needs configuration, we have already covered this part.

If Joe didn't already have the PptViewer application, the current implementation of SDM does not provide many options for Joe to retrieve the application. The only way of installing new software, is through the SDM's API.

---

4. *When Eve tries to configure the PptViewer on Joe's UbiNode to use the room's projector,* **the UbiNode alerts** *"No access to resource!".*

Not Implemented: No such alert system currently exists in the SDM. It is currently not possible to make this kind of alerts unless the configuration mentioned in the scenario was accessed through a separate GUI accessible only from the PptViewer.

---

5. *Eve has forgotten the username and password she received for the projector, so instead, she uses her UbiNode to* **share the projector with Joe**.

Not implemented: The SDM does not allow users to share Proxy Services through a GUI. Sharing needs to be done on the API level.

---

6. *Finally Joe is able to configure his PptViewer to use the shared projector instead of using it directly.*

Not implemented: This may be possible, dependent on how the PptViewer interprets its configuration. Still, SDM does not directly support such an action. An example of how that would look like. would be a configuration in which the user could select the output device of the application, changing from for instance "Projector" to "Projector (shared by Eve)".

6.4.1.1 Summary

Some of the points above we have already identified in earlier evaluations. SDM covered the basic functionality that were needed in the scenario (viewing a presentation on a projector).

However, some new capabilities were discovered that the SDM lacks.

The scenario tried to alert the user when a Proxy Service was configured wrong (wrong username and/or password), but this is not currently possible. In general, the configuration should support more dynamic content.

Also, Joe had the need to switch the output of the application to another Proxy Service than it was currently set to. Connecting resources with each other is a crucial capability in ubiquitous computing, which SDM does not currently support.

6.5 Research questions

This section recapitulate and states contributions to the research questions stated in section 1.3.

6.5.1 RQ-1

RQ-1: *How can we extend existing service management architectures to support user-centered and community-based service management?*

The starting point of this thesis was the Service Domain Manager (SDM), implemented by Johansen [3] and improved by Mora [4]. This work has been extended by the following contributions:

- Suggesting an improved version of the SDM, which aims to provide user-centered and community-based service management. This work is presented in Chapter 4 and serves as basis for evaluation of the implementation.
- Implementing parts of the suggested improvement of the SDM. This work is presented in Chapter 5 and evaluated in sections 6.2 and 6.2.1.

The improved SDM adds support for user-centered service-management in the following way:

- A GUI has been implemented for managing Proxy Services.
- The user can list installed Proxy Services in various categories, which are based on which Collaboration Space they belong to, which type they are of, the location of the user, and other categories.
- Supporting configuring of Proxy Services.
- Supporting sharing of Proxy Services among users.
- Supporting starting, stopping and uninstalling of Proxy Services.
- The user can view the description of a Proxy Service.

The improved SDM adds support for community-based service-management by:

- Supporting sharing of Proxy Services among users.

6.5.2 RQ-2

RQ-2: *What technologies, architectures and platforms are most suitable for implementing user-centered and community-based service management?*

The contribution to this research question is based on the work done in Chapter 3, "State of the art", and Chapter 5, "Implementation".

The following four items were found suitable:

1. Distribution platforms for mobile applications, such as Apple's App Store and Google's Android Market, because they provide a suitable model for how software could be deployed on a user's mobile device.
2. OSGi, because the platform allows dynamic deployment and management of resources and software.
3. R-OSGi, because it provides a suitable framework that enables users to share resources and software functionality with other users.
4. HTTP-based communication using Java Servlets, because it made it possible to create a simple solution for enabling users to share resources and software functionality with other users.

6.5.2.1 Rationale

Some additional reasoning are made for some the selections above:

1. By "suitable model", we mean the way users retrieve applications using a distribution platform for mobile applications, as described in section 3.2.1.4.
2. Although OSGi has been used in earlier work of Ubicollab, it is included in the list above because it has not been used with the purpose of creating user-centered and community-based service management on mobile devices. OSGi allows dynamic deployment as described in section 3.1.4 and management of resources and software as described in section 4.2.2. OSGi in itself does not directly support user-centered service management, but it's API makes it easy to implement this. The "community-based" service management aspect is not covered by OSGi.
3. R-OSGi enables user-centered and community-based service management in that it allow transparent use of remote resources through Proxy Services. Even though not used in our solution because of technical difficulties, the solution were found to be suitable.
4. Enabling sharing of resources and software functionality is a cornerstone of community-based service-management.

6.5.3 RQ-3

RQ-3: *How can we evaluate the usability and utility of user-centered and community-based service management? What are the most compelling scenarios?* The SDM represents a solution to user-centered and community-based service management, as stated in section 6.5.1, and can thus be used as the subject for evaluation.

6.5.3.1 Evaluating usability

Evaluation of usability has been suggested for further work, see section 7.3.

6.5.3.2 Evaluating utility

This thesis has evaluated the utility of the SDM, by using the three methods presented in section 6.1.

Method 1: Examining coverage of requirements specification

The fulfillment of the SDM's requirements specification was examined to get an overview the SDM's completeness. Section 6.2 stated that the functions implemented were related to management and sharing of software

components, while functions partially or not implemented were related to the discovery of resources.

Method 2: Create scenarios that demonstrate problems and their solutions, and compare the SDM's functionality with that of the solutions

Next, the evaluation of the SDM's utility was done in the following way:

1. A realistic, detailed scenario, named the ProblemScenario, had already been made to elicit challenges with today's technology and solutions. A system was proposed to solve the challenges in this scenario. The utility of this proposed system was demonstrated through a new scenario, named the SolutionScenario, which described how the system solved the problems from the ProblemScenario.
2. The SDM attempted to implement the proposed system. Because the proposed system's utility already was demonstrated through the SolutionScenario, the SDM's utility could be evaluated by comparing its implemented functionality with the proposed system. This comparison was conducted in section 6.2.1.

Method 3: Demonstrating utility through one or more real-world, proof-of-concept applications.

As for method 3 in section 6.1, the PptViewer has been evaluated, but it utilizes only parts of the SDM's capabilities related to sharing and configuration of resource controllers. Nonetheless, the parts that were evaluated did demonstrate SDM's utility of enabling users to share resource controllers with other users, and configure resource controllers. This was done by demonstrating how the PptViewer was used to solve challenges in a realistic scenario. The evaluation of the PptViewer was conducted in section 6.4.

Our contribution to RQ-3

Our contribution to RQ-3 are the generalization of methods 1 to 3, described above. We can evaluate the utility of user-centered and community-based service management systems, by doing the following steps:

1. Get an understanding of the system's completeness by examining its fulfillment of its requirements specification.

2. Create a scenario that describe current problems. Suggest and describe a system that solves these problems. Demonstrate the suggested system's utility by creating a scenario in which the problems are solved by the system. Then compare the suggested system's functionality with that of the existing service-management system.
3. Create one or more real-world, proof-of-concepts applications that demonstrate the utility of the service-management system.

6.5.4 Future work: Selection of scenarios

No research has been done on how to create the most compelling scenarios. This thesis used a problem-solution pair of scenarios, in which the solution scenario was a response to the problem scenario. This may approach may have made the scenarios more compelling than making a problem-solution scenario pair without any relation.

In any case, this research is left for future work.

6.6 Limitations of this thesis

Privacy and security

Privacy and security of the users have been not been addressed in this thesis, and this constitutes a major limitation of this thesis. See section 7.3, “Further work” for further details.

Lack of evaluation of usability

The SDM’s usability has not been evaluated, because of time constraints, and to some extent, incompleteness of the SDM's implementation. The SDM's functionality is limited, which makes it challenging to make realistic experiments to test for usability. Evaluation of usability for service-management systems in general remains for further work, see section 7.3.

A GUI for retrieving software

The implementation of the SDM does not provide a GUI for retrieving appropriate software when discovering resources, so it is difficult to test and evaluate realistic usage of the SDM.

Integration with the existing implementation of Ubicollab

The SDM has not been tested with the existing implementation of Ubicollab, which was last updated by Mora [4], which will make future work with the Ubicollab code harder.

Further, package names, interfaces and method signatures have been changed with the purpose of improving the current system, but with the negative side-effect that the existing implementation of Ubicollab must be updated. The coupling between the SDM and other components of Ubicollab were, however, investigated, and there were only dependencies to the SDM from the existing Resource Discovery Manager.

Other limitations

The implementation had other technical limitations as well, but are of less importance for the work of this thesis, and may be more of interest for future developers. These limitations are presented in Appendix C.6.

Chapter 7 Conclusion and further work

This chapter summaries what has been done in this thesis, and lists the contributions made. The structure are as follows:

- Section 7.1 summaries the work of this thesis.
- Section 7.2 describes the contributions made.
- Section 7.3 lists suggestions for further work.

7.1 Summary

In a ubiquitous environment, users continuously encounter new resources that can enhance the user experience. As users encounter an increasing number of resources, the management of these resources becomes a central task.

This thesis has used a scenario-driven approach to identify challenges related to user-centered and community-based management of resources in a ubiquitous environment. Challenges identified include

1. difficulties with retrieving appropriate software for utilizing a resource,
2. increased resource consumption and how users can handle this, and
3. how to share resources among users.

These challenges were used as basis for creating a high-level requirements specification for a system that overcomes the identified problems.

Next, we examined the state-of-the-art related to the identified challenges in order to find ideas and solutions for addressing the identified problems. These ideas and solutions were used as basis for suggesting a system that intends to solve these problems. The suggested system demonstrated its ability to solve the problems through a scenario.

The suggested system, named Service Domain Manager (SDM), has been implemented on the Google Android platform. The SDM do not solve problem one in the list above because of lack of time for implementation. The SDM addresses mainly problem two and three in the list above. A demonstration application was also implemented to show that the SDM solves problem three above. An evaluation of the SDM was done that included contributions to the research questions of this thesis. These contributions are presented in section 7.2.

7.2 Contributions

7.2.1 Contribution 1

RQ-1: *How can we extend existing service management architectures to support user-centered and community-based service management?*

We have implemented the Service Domain Manager (SDM), which is a system running on the OSGi platform that adds support for user-centered service-management by including features for managing services on the Ubicollab platform. The SDM provides a GUI that enable the user to:

- List services by using different, partly context-based, categories.
- Configure services.
- Start, stop or uninstall services.
- View service's description.

Additionally, the SDM provides an API that makes it possible to share services among users across the network.

7.2.2 Contribution 2

RQ-2: *What technologies, architectures and platforms are the most suitable for implementing user-centered and community-based service management?*

The following four items were found suitable:

1. Distribution platforms for mobile applications, such as Apple's App Store and Google's Android Market, because they provide a suitable model for how software could be deployed on a user's mobile device.
2. OSGi, because the platform allows dynamic deployment and management of resources and software.

3. R-OSGi, because it provides a suitable framework that enables users to share resources and software functionality with other users.
4. HTTP-based communication using Java Servlets, because it made it possible to create a simple solution for enabling users to share resources and software functionality with other users.

7.2.3 Contribution 3

RQ-3: *How can we evaluate the usability and utility of user-centered and community-based service management? What are the most compelling scenarios?* We can evaluate the utility of user-centered and community-based service management systems, by doing the following steps:

1. Get an understanding of the system's completeness by examining its fulfillment of its requirements specification.
2. Create a scenario that describe current problems. Suggest and describe a system that solves these problems. Demonstrate the suggested system's utility by creating a scenario in which the problems are solved by the system. Then compare the suggested system's functionality with that of the existing service-management system.
3. Create one or more real-world, proof-of-concepts applications that demonstrate the utility of the service-management system.

7.2.4 Other contributions

This thesis is itself a contribution to the Ubicollab project, because we suggest a system for the Service Domain Manager, which is a component of the Ubicollab platform [2].

Further, while the previous sections describe research contributions, an additional contribution was made as well:

A Java package for building graphical user interfaces (GUI) in OSGi. The package can be used to build HTML based-GUIs in OSGi based on Java Servlets. It is deployed as an OSGi bundle, but can be used in non-OSGi based applications as well.

7.3 Further work

How can we support privacy and security of users?

Solutions that aims to provide community-based service management and ubiquitous collaboration in a mobile environment face serious challenges for supporting privacy and security for users. This thesis is a work in this area, but these challenges has not been addressed, and this constitutes a major limitation of this work.

Some challenges related to privacy and security for users that needs further research are:

- When new software is needed for utilizing discovered resources, how can we avoid retrieving and running malicious code?
- How can we minimize the resource consumption when securing data in mobile, peer-to-peer based systems?
- When users share data in their context with other users, how can we avoid that this data is read, altered or disrupted by an adversary?

Related research to these challenges are authentication, certificates, identity management, access control, encryption, secure peer-groups and trusted sources.

Evaluation of usability

Parts of RQ-3 questioned for how usability can be evaluated for user-centered and community based service management, and what the most compelling scenarios are. These questions has not been addressed by this thesis, and is thus left for future work. Appendix A contains two suggested usability experiments that can be conducted. Finding compelling scenarios will also be a part of this

Make R-OSGi work on Android

R-OSGi is a feature-rich and high-performing OSGi bundle that enables the creation of distributed applications and sharing of resources and functionality. It appears to be a suitable approach for Ubicollab, though this must be

investigated further. However, we did not succeed in making R-OSGi work on the Android platform.

Glossary

Resource	A resource, or system resource, is a physical or non-physical component of limited availability. Devices connected to a computer system or communication network are physical resources [3]. Internal system components are physical resources. Examples of non-physical system resources include files, network connections and services.
Application	Software that allows a user to perform tasks through a GUI.
Device driver	Gives access to a hardware device on a computer. Usually used by higher-level software for accessing some hardware device. The term is sometimes referred to by replacing "Device" with the actual hardware, for instance "printer driver", which means the device driver for a printer.
Resource controller	Software that can access a resource through its API. The equivalent on the Ubicollab platform is a Proxy Service. A resource controller is also similar to a device driver.
Proxy Service	Proxy Services implements a uniform way of communicating with resources [2], similar to device drivers and resource controllers. They can also be applications that implement a GUI and utilize other Proxy Services.
Software component	Either a resource controller or an application.
UbiNode	A mobile device in the Ubicollab network that has installed the Ubicollab platform, and provides or uses Proxy Services [3]. Ubicollab users have their Proxy Services installed on UbiNodes.
Collaboration Instance	Provides a shared context for users, in which users can put data. It can represent an activity, a social world, a locale, a cognitive system, or a setting [2].
Bundle	A JAR file used by in the OSGi platform to implement functionality.
ProblemScenario	A reference to the scenario in section 2.1.
SolutionScenario	A reference to the scenario in section 4.1.
Service Oriented Architecture (SOA)	A perspective of software architecture that defines the use of services to support the requirements of

	users. In a SOA environment, resources on a network are made available as independent services that can be accessed without knowledge of their underlying platform implementation. In SOA the interface definition hides the implementation of the language-specific service. SOA-compliant systems can be independent of development technologies and platforms.
Ubicollab platform	The Ubicollab platform is the container onto which Proxy Services are deployed, and it consists of core components such as resource discovery, service management, and collaboration management. In addition this.
Usability	Usability refers to *"the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use*" [47].
OSGi service	"An OSGi service is a java object instance, *registered* into an OSGi framework with a set of *properties*. Any java object can be registered as a service, but typically it implements a well-known interface" [48]. OSGi bundles can register zero or more OSGi services, which other bundles or the bundle itself can retrieve. As a result, java object instances can be accessed across bundles.
Service	A service is a program that can run in the background, and is intended for longer-running operations or for providing functionality to applications.

References

[1] M. Weiser, “The Computer for the 21st Century,” *Scientific American*, Sep. 1991.

[2] B.A. Farshchian og M. Divitini, “Collaboration Support for Mobile Users in Ubiquitous Environments,” *Handbook of Ambient Intelligence and Smart Environments*, Heidelberg: Springer, 2009, s. 173-199.

[3] K. Johansen, “User-centered and collaborative service management in UbiCollab - Design and implementation,” Master, Norwegian University of Science and Technology (NTNU), 2007.

[4] S. Mora, “A mobile extensible architecture for implementing ubiquitous discovery gestures based on object tagging,” Master, Norwegian University of Science and Technology, 2009.

[1] A.R. Hevner, S.T. March, J. Park, og S. Ram, “Design Science in Information Systems Research,” *MIS Quarterly*, vol. 28, 2004, s. 75–105.

[1] J.G. Daugman, “High confidence visual recognition of persons by a test of statistical independence,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 15, Nov. 1993, s. 1148 -1161.

[7] S.B. Lee og S. Tsutsui, *Intelligent biometric techniques in fingerprint and face recognition*, Boca Raton, FL, USA: CRC Press, Inc., 1999.

[8] “Android.com - Android at Google I/O.” http://www.android.com/.

[9] “Apple - iPhone 4 - Video calls, multitasking, HD video, and more.” http://www.apple.com/iphone/.

[10] “UPnP Forum.” http://www.upnp.org/.

[11] “Home - DLNA.” http://www.dlna.org/home. http://www.osgi.org/Main/HomePage.

[12] “OSGi Alliance | Main / OSGi Alliance.” http://www.osgi.org/Main/HomePage.

[13] W. Woerndl, C. Schueller, og R. Wojtech, “A Hybrid Recommender System for Context-aware Recommendations of Mobile Applications,” *ICDEW '07: Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering Workshop*, Washington, DC, USA: IEEE Computer Society, 2007, s. 871–878.

[14] J.Y. Jung og J.W. Lee, “Automatic Discovery and Installation of Wearable Bio Signal Devices in Ubiquitous Healthcare System,” *Advanced Communication Technology, The 9th International Conference on*, 2007, s. 412 -414.

[15] “Microsoft Tag - Connecting Real Life and the Digital World using Mobile Barcodes!.” http://tag.microsoft.com/consumer/index.aspx.

[16] “NeoReader: Home.” http://www.neoreader.com/.

[17] “List of digital distribution platforms for mobile devices - Wikipedia, the

free encyclopedia."
http://en.wikipedia.org/wiki/List_of_digital_distribution_platforms_for_mobile_devices.
[18] "148Apps.biz | Apple iTunes App Store Metrics, Statistics and Numbers for iPhone Apps." http://148apps.biz/app-store-metrics/.
[19] "Ubuntu Wins Most User Friendly Linux Distribution Award | Ubuntu." http://www.ubuntu.com/news/MostUserFriendlyAward.
[20] OSGi Alliance, "RFC-0112 Bundle Repository," 2005. http://www.osgi.org/About/FAQ.
[21] "Advanced Task Manager v4.1 Application for Android | Tools." http://www.androlib.com/android.application.com-arron-taskmanager-zpp.aspx.
[22] M. Matsumoto, R. Kiyohara, H. Fukui, M. Numao, og S. Kurihara, "Proposition of the context-aware interface for cellular phone operations," *Networked Sensing Systems, 2008. INSS 2008. 5th International Conference on*, 2008, s. 233 -233.
[23] D. Kamisaka, S. Muramatsu, H. Yokoyama, og T. Iwamoto, "Operation Prediction for Context-Aware User Interfaces of Mobile Phones," *Applications and the Internet, 2009. SAINT '09. Ninth Annual International Symposium on*, 2009, s. 16 -22.
[24] E. Valavanis, C. Ververidis, M. Vazirgianis, og G.C. Polyzos, "MobiShare: Sharing Context-Dependent Data and Services from Mobile Sources," *In Proceedings of IEEE/WIC International Conference on Web Intelligence (WI'03*, 2003, s. 13–17.
[25] J. Rellermeyer, G. Alonso, og T. Roscoe, "R-OSGi: Distributed Applications Through Software Modularization," *Middleware 2007*, 2007, s. 20, 1.
[26] "OSGi Alliance | Release4 / Download Specifications." http://www.osgi.org/Release4/Download.
[27] "Introduction to Test Driven Design (TDD)." http://www.agiledata.org/essays/tdd.html.
[28] "OSGi Alliance | About / FAQ." http://www.osgi.org/About/FAQ.
[29] N. Bartlett, "OSGi in Pracice," *OSGi in Pracice*, s. 1-13.
[30] "comScore Reports May 2010 U.S. Mobile Subscriber Market Share - comScore, Inc." http://www.comscore.com/Press_Events/Press_Releases/2010/7/comScore_Reports_May_2010_U.S._Mobile_Subscriber_Market_Share.
[31] "What is Android? | Android Developers." http://developer.android.com/guide/basics/what-is-android.html.
[32] C. Kindel, "Different Means Better with the new Windows Phone Developer Experience - Charlie Kindel on Windows Phone Development - Site Home - MSDN Blogs."
[33] "ProSyst - Developer Zone." https://dz.prosyst.com/devzone/Mobile.

[34] “Eclipse.org home.” http://www.eclipse.org/.
[35] “OSGi Android - open source server - Confluence.” http://opensource.luminis.net/wiki/display/SITE/OSGi+Android.
[36] “embedded Rich Client Platform (eRCP).” http://www.eclipse.org/ercp/.
[37] “About the JFC and Swing (The Java™ Tutorials > Creating a GUI With JFC/Swing > Getting Started with Swing).”http://download.oracle.com/docs/cd/E17409_01/javase/tutorial/uiswing/start/about.html.
[38] “User Interface | Android Developers.” http://developer.android.com/guide/topics/ui/index.html.
[39] “android.webkit | Android Developers.” http://developer.android.com/reference/android/webkit/package-summary.html.
[40] “UIWebView Class Reference.” http://developer.apple.com/iphone/library/documentation/uikit/reference/UIWebView_Class/Reference/Reference.html.
[41] “Java Servlet Technology.” http://java.sun.com/products/servlet/.
[42] “PHP: Hypertext Preprocessor.” http://php.net/.
[43] “JavaServer Pages Technology.” http://java.sun.com/products/jsp/.
[44] “WebView | Android Developers.” http://developer.android.com/reference/android/webkit/WebView.html.
[45] “SourceForge.net: Topic: R-OSGi for Android?.” http://sourceforge.net/projects/r-osgi/forums/forum/533562/topic/3220253/index/page/1.
[46] W.K. Edwards, V. Bellotti, A.K. Dey, og M.W. Newman, “The challenges of user-centered design and evaluation for infrastructure,” *Proceedings of the SIGCHI conference on Human factors in computing systems*, Ft. Lauderdale, Florida, USA: ACM, 2003, s. 297-304.
[47] “ISO 9241-11:1998 - Ergonomic requirements for office work with visual display terminals (VDTs) -- Part 11: Guidance on usability.” http://www.iso.org/iso/catalogue_detail.htm?csnumber=16883.
[48] “OSGi Service tutorial.” http://www.knopflerfish.org/osgi_service_tutorial.html#what_is.
[49] “Discover the secrets of the Java Serialization API.” http://java.sun.com/developer/technicalArticles/Programming/serialization/.

Appendix A: Usability experiment suggestions

A.1 Usability experiment 1: Viewing a presentation

The PptViewer demonstration application (see section 5.5.7) can be used as basis for a simple experiment with users. The experiment's goal is to evaluate the usability of the SDM's configuration capability. Users will get the task to view a presentation using PptViewer, then open its configuration and change the URL to the host web application, and then view the presentation again to confirm that it is working.

A.1.1 Test plan

The prerequisites for the experiment are the following:

- Users must have access to a mobile device with the SDM installed, as well as the demonstration application PptViewer.
- A PPT (Microsoft PowerPoint) file must be placed on a location that the user's mobile device can access via a URL.
- A web server with PHP support.
- The web application for PptViewer must reside two places on the web server. That is, there should exist two unique URLs that point to the web application, for instance: *http://<serverIp/pptViewer* and *http://<serverIp>/anotherPptViewer*.
- The PptViewer client application must be configured to use one of the URLs to the web application, for instance *http://<serverIp/pptViewer.*

Users will receive the following task:

"

Task 1: Your first task is to view a presentation on a projector. The URL to the presentation is [insert a valid URL to a PPT file here]*. Open the PptViewer, and use the PptViewer to view the slides in this presentation.*

Task 2: Your next task is to configure the PptViewer to use another web host application. Do this by starting the Service Domain Manager, and then opening the configuration for the PptViewer service. Change the URL of the web host to http://<serverIp>/anotherPptViewer.

Now repeat Task 1 and confirm that you can still view a presentation.

"

A.2 Usability experiment 2: Managing Proxy Services

The user's mobile device can be pre-installed with many Proxy Services. These Proxy Services can be configured to reside in a various pre-defined Collaboration Spaces (CIs), and have their type attribute set to various names as well.

The experiment's goal is to evaluate the usability of the SDM's service management capabilities, or more specifically, the functions "stop service" and "uninstall service" (available from the GUI).

The users will get the task to find certain Proxy Services and either stop or uninstall them. The users will have to browse the categories in order to find the Proxy Services, and then uninstall or stop the Proxy Service.

Appendix B: Contributions

Additionally, the work listed above has resulted in additional contributions that are not directly related to the user experience of collaboration setup process, but serve as concepts or tools for aiding such work.

- **A Java package for building graphical user interfaces (GUI) in OSGi.** The package can be used to build HTML based GUIs in OSGi based on Java Servlets. This can be used by OSGi developers, for instance for further development of the Ubicollab platform.

Appendix C: Implementation details

C.1 Ubicollab architecture

The architecture of Ubicollab is shown in figure .

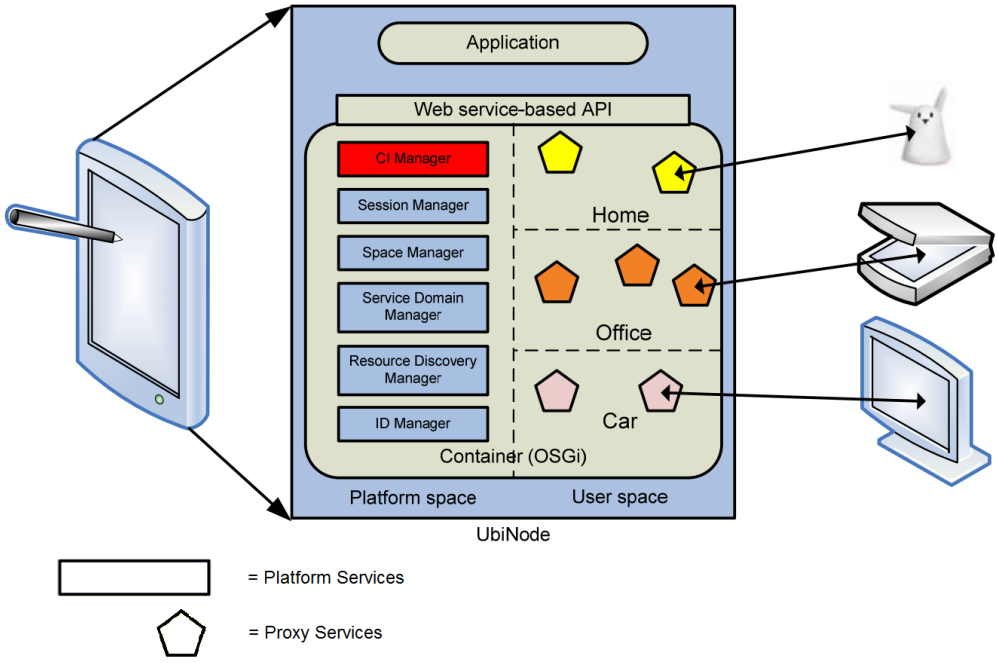


Figure C.1: The architecture of the Ubicollab platform.

C.2 Mapping of figure names to real package names

Name from Figure 5.3	Real Java package name
SpaceManager	sm.model
ProxyService	core.service
Proxy	core.service.proxy
ViewServlet	view.servlet
SDM Core	core.sdm
SDMServlet	core.sdm.view.servlet
ViewAndroid	core.sdm.view.android
JUnit	org.knopflerfish.bundle.junit
JUnit	org.knopflerfish.bundle.junit_runner
SDM Test	testing.sdm.junit

Table C.1: Mapping of Java package names.

C.3 Selecting GUI technologies

This section describes the considerations made when selecting the GUI technologies.

C.3.1 An HTML-based GUI

Described in section5.3.4.1.

C.4 Identifying an OSGi bundle as a Proxy Service

C.4.1 Required properties

Proxy Services are implemented using OSGi bundles, but have in this implementation been given additional properties so that the Ubicollab platform can separate them from regular OSGi bundles. These properties as described as follows.

- The `Bundle-SymbolicName` field in the `MANIFEST.MF` file of an OSGi bundle must start with the text `org.ubicollab.service`.

For instance, a Proxy Service in SDM's implementation named PptService uses the following line in its `MANIFEST.MF` file:

```
Bundle-SymbolicName: org.ubicollab.service.PptService
```

Listing C.1: The manifest-entry required for identifying an OSGi bundle as a Proxy Service.

- The Proxy Service must register a class implementing the interface `ServiceLink` as a OSGi service. A ServiceLink is used to communicate with the Proxy Service.

C.4.2 Optional type field

In order to categorize Proxy Services by type (see section), a Proxy Specify which

A Proxy Service has an optional attribute called *type*. Setting a Proxy Service's type allows it to be put into a category corresponding to that type, as shown in Figure 5.11.

In order to set the *type* of a Proxy Service, a new field, `Ubicollab-Service-Type`, must be set in the bundle's `MANIFEST.MF` file. For instance, to set the type to "display ppt", the manifest field is set the following way:

```
Ubicollab-Service-Type: display ppt
```

Listing C.2: The manifest-entry for setting the type of a Proxy Service.

If not type field is set, the Proxy Service is not associated with any type.

C.5 Sharing of Proxy Services

The detailed message flow between two users' UbiNodes in a sample scenario are shown in Figure C.2. The scenario involves the usage of the PptViewer application presented in section 5.5.7.

C.5.1 Internal message flow

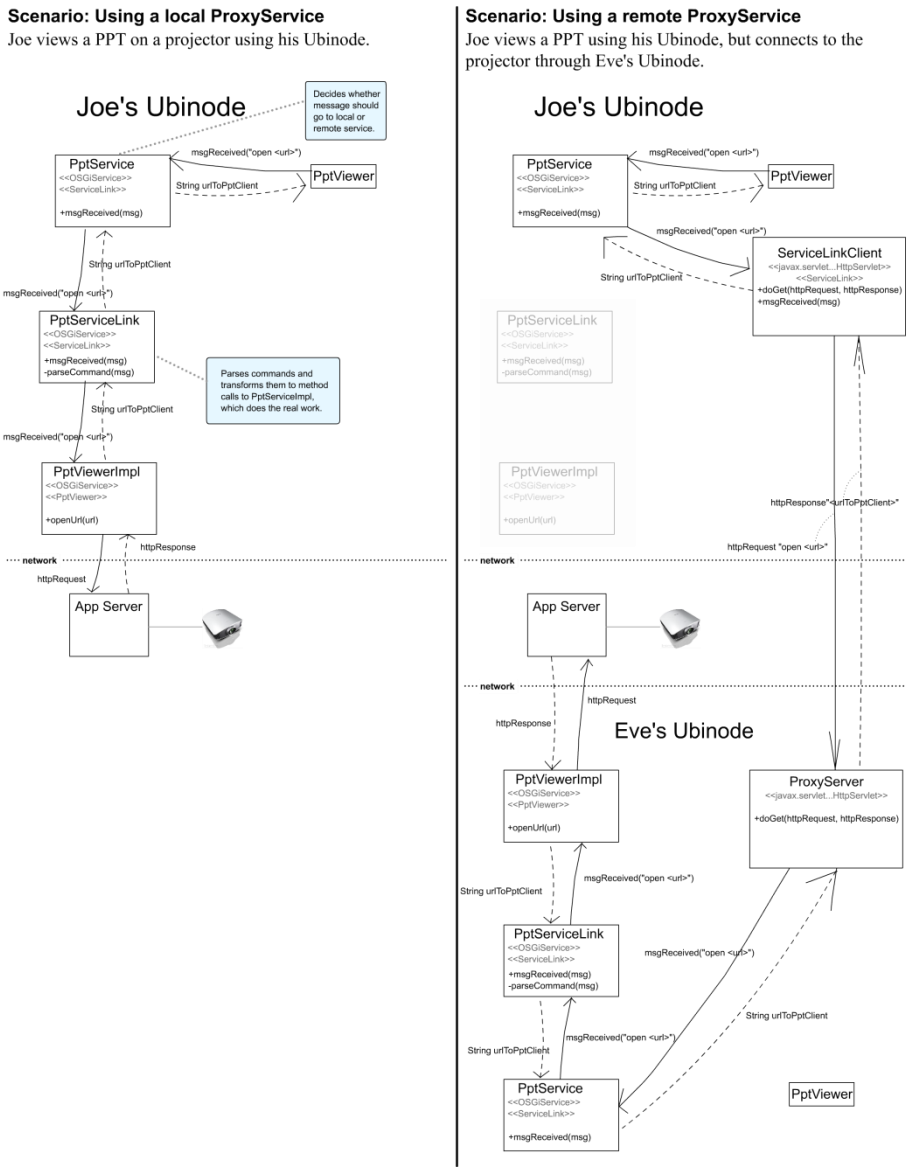


Figure C.2: The message flow when using a shared Proxy Service.

C.6 Known issues with current implementation

C.6.1 Limited capabilities of sharing of Proxy Services

One of the capabilities of the SDM is the possibilities for a user to share Proxy Services with other users. However, this possibility is very limited, because:

- Using a shared Proxy Services can only be used by sending text messages as Java String instances. However, Java Serialization [49] should be investigated in order to address this limitation.
- Because messages are sent by setting a GET-parameter of an URL to the shared Proxy Service, the length of the message is limited to the size of that GET-parameter.

C.6.2 No concurrency handling

Concurrency handling of for instance Java Collections have not been dealt with. This is needed especially for the components involved in the sharing of Proxy Services.